871780 — MonB5G — ICT-20-2019-2020

Deliverable D4.1
Initial Report on AI-Driven Techniques for the MonB5G Decision Engine

Document Summary Information

Grant Agreement No	871780	**Acronym**	MonB5G
Full Title	Distributed Management of Network Slices in beyond 5G		
Start Date	01/11/2019	**Duration**	36 months
Project URL	https://www.monb5g.eu/		
Deliverable	D4.1 – Initial report on AI-driven techniques for the MonB5G Decision Engine		
Work Package	WP4		
Contractual due date	M17	**Actual submission date**	31.03.2021
Nature	Report	**Dissemination Level**	Public
Lead Beneficiary	EUR		
Responsible Author	Thrasyvoulos Spyropoulos (EUR), Theodoros Giannakas (EUR)		
Contributions from	Theodoros Giannakas (EUR), Thrasyvoulos Spyropoulos (EUR), Pavlos Doanis (EUR), Marina Costantini (EUR), Christos Verikoukis (CTTC), Hatim Chergui (CTTC), David Pubill (CTTC), Jordi Serra (CTTC), Luis Blanco (CTTC), Sarang		

	Kahvazadeh (CTTC), Luis A. Garrido Platero (IQU), Anestis Dalgkitsis (IQU), Lanfranco Zanzi (NEC), Francesco Devoti (NEC), Jose Jurandir Alves Asteves (ORA-FR), Amina Boubendir (ORA-FR), Sławomir Kukliński (ORA-PL), Robert Kołakowski (ORA-PL), Sihem Bakri (EUR), Bouziane Birk (EUR), Adlen Ksentini (EUR), Anne-Marie Bonseag (LMI), Mohamed Rahali (BCOM), Cao Thanh Phan (BCOM), Zhao Xu (NEC), Aiman Nait Abbou (AAL), Mohammed Boukhalfa (AAL)

TABLE OF CONTENTS

List of Figures

List of Tables

List of Acronyms

Acronym	*Description*
3GPP	*Third Generation Partnership Project*
AE	Analytic Engine
AE-F	Analytic Engine Function
AE-S	Analytic Engine Sublayer
AI	Artificial Intelligence
CLA	Closed-loop Automation
CNF	Cloud Native function
DE	Decision Engine
DE-F	Decision Engine Function
DE-S	Decision Engine Sublayer
EEM	Embedded Element Manager
eMBB	Enhanced Mobile Broadband
eTOM	Enhanced Telecom Operations Map
ETSI	European Telecommunications Standards Institute
ECA	Event Condition Action
ENI	Experiential Networked Intelligence
FCAPS	Fault, Configuration, Accounting, Performance, Security
ILP	Integer Linear Program(ing)
ISM	In-Slice Management
ITU	International Telecommunication Union
KPI	Key Performance Indicator
LCM	Lifecycle Management
ML	Machine Learning
MANO	Management and Orchestration
MaaS	Management as a Service
MAN-F	Management Function
mMTC	Massive Machine Type Communications
MEO	MEC Orchestrator
MNO	Mobile Network Operator
MLaaS	MonB5G Layer as a Service
MS	Monitoring System
MS-F	Monitoring System Function
MS-S	Monitoring System Sublayer
MEC	Multi-access Edge Computing
NFVO	Network Function Virtualization Orchestrator
NSD	Network Service Descriptor

NSO	Network Service Orchestrator
NSP	Network Service Provider
NSI	Network Slice Instance
NSMF	Network Slice Management Function
NSSMF	Network Slice Subnetwork Management Function
NST	Network Slice Template
NSSI	Network sub-Slice Instance
NGMN	Next Generation Mobile Networks
NFVI	NFV Infrastructure
OAI	Open Air Interface
ONAP	Open Network Automation Platform
OSM	Open Source MANO
OSS	Operation System Support
PaaS	Platform as a Service
PNF	Physical Network Function
PoC	Proof of Concept
QoE	Quality of Experience
QoS	Quality of Service
RL	Reinforcement Learning
RAN	Radio Access Network
SDN	Software-Defined Networking
SON	Self-Organizing Network
SLA	Service Level Agreement
SFL	Slice Functional Layer
SML	Slice Management Layer
SM	Slice Manager
uRLLC	Ultra-Reliable Low-Latency Communication
VIM	Virtual Infrastructure Manager
VNF	Virtual network Function
VNFM	Virtual network Function Manager
ZSM	Zero-touch network and Service Management

1 Executive summary

This deliverable describes our preliminary work related to WP4, namely data-driven and distributed orchestration mechanisms for slice admission control, intra-slice, and inter-slice management. The key avenues of innovation for the mechanisms proposed in WP, the initial progress of which we describe here, are the following.

First, WP4 proposes *a fully data-driven decision engine and algorithm stack.* While recent works and related EU projects investigate the introduction of AI solutions into the beyond 5G architecture, these either focus mostly on data analytics, or target specific components of the orchestration architecture in a piecemeal fashion. In contrast, WP4 investigates data-driven algorithms as the key ingredient of all slice lifecycle management operations, from admission control, to scaling and migration operations targeting a specific slice (we refer to these as *intra-slice* operation) and/or specific technological (e.g., RAN, MEC, core, cloud) and administrative domain, to end-to-end orchestration operations involving large number of slices each spanning multiple technological and possibly administrative domains. This is greatly facilitated by a tight integration with an equally novel Analytic Engine (AE) and Monitoring System (MS), also heavily based on modern AI methods, that feed the algorithms of the Decision Engine (DE) with multi-grain, multi-modal, and tunable measurements and predictions.

Second, another key novelty of MonB5G and the mechanisms of WP4 is that the DE architecture is designed to be flexibly distributed across different administrative domains, technological domains, and even fine granularities such as one DE per tenant, slice, or even VNF (virtual network function). Flexible distribution means that the proposed DE could operate in different configurations, depending on the need. For example, a *hierarchical DE* implementation could be useful, where a "central" DE (e.g., running in the cloud or core domain) is in control of various "edge" DEs (partially) responsible for different RAN domains, edge clouds (e.g., MEC) etc., (or even slice-specific DEs, as mentioned earlier) all working towards optimal lifecycle management of slices spanning all these domains. On the one hand, having the DE algorithms located as close as possible to the network components that the decisions are taken for, can greatly improve decision latency, a key concern when controlling edge resources (e.g., RAN) where even management decisions often need to be taken at much smaller time granularities than traditional in traditional management frameworks. On the other hand, the amount of information that needs to be collected and transported across the network to facilitate the heavy use of modern data-driven algorithms will be orders of magnitude more than in traditional monitoring and analytics for network management. What is more, if the trained model used is large (e.g., state-of-the-art deep neural network architectures used for challenging ML tasks might have millions of trained weights – and thus resulting operations). Hence, in this deliverable, we will also present mechanisms that not only distribute the conceptual architecture across domains (e.g., the same DNN model running with no or minor modifications in different locations) but *distribute the model/algorithm itself across domains, such Distributed Deep Neural Networks*

Lastly, in this deliverable we will also consider solutions that fully decentralize the DE, across domains, in a non-hierarchical, flat topology, without a "central" entity coordinating the rest. This setup is often referred to as *Federated Learning.* In beyond 5G networks, federated DEs are often applicable in scenarios where a slice spans multiple administrative domains, and the local DE of one domain needs to coordinate with other DEs to optimize a common goal (e.g., the end-to-end performance/SLA of a slice spanning these domains), yet it does not want to reveal any data to other DEs, besides what is absolutely necessary, for privacy reasons.

It is important to remark the following here: (a) while proposals along each of the above directions has standalone value, we argue that it is the co-design of these mechanisms that makes the sum more than its parts, and a key novelty of the WP4 mechanisms; (b) while the DE engine architecture will eventually cover all technological domains an multiple administrative ones, some of the initial algorithms explored in the first phase of the project, and described in detail in this deliverable, target specific domains as a key initial step towards an integrated end-to-end multi-domain setup. These algorithms should be interpreted as building blocks of the full integrated architectures that the proposed DE will evolve to in the remainder of the project.

The deliverable content flow can be summarized as follows. We commence our exposition with a generic description of the types of problem setups considered in this work, with respect to key components the proposed algorithms will consider, namely slice representations, KPI metrics for the algorithms to optimize as well as KPIs used to measure the performance of the algorithms themselves, SLA types and violation penalties, etc. We then provide an up-to-date state-of-the-art discussion on existing solutions for problem falling under the umbrellas of data-driven orchestration, zero-touch slice management, etc., coming both from the academia as well as industrial white papers, standards, etc. We focus on the key tools that have attracted significant interest recently, and used by our own algorithmic frameworks, as well, namely (deep) reinforcement learning, use of deep neural networks for slice-specific objectives, etc. With this setup in mind, we move on to describe the architecture of the (distributed) Decision Engine proposed, identifying key component, interaction between DEs residing in the same or different domains, as well as the interfaces with the Monitoring System (MS) and Analytic Engine (AE).

The rest of the deliverable chapters focus on algorithmic contributions for specific components of slice management, namely slice admission control, intra-slice orchestration, and inter-slice orchestration; these topics correspond respectively to the three technical tasks of the work package. Specifically, we first describe initial proposals and findings for slice admission control using data-driven methodology (e.g., reinforcement learning) and discuss how modern admission control mechanisms for the scenarios of interest should also take into consideration future reconfiguration costs (due to new admissions, or demand variability and resulting resource scaling), as well as generally interact with inter- and intra- slice management mechanisms active during the lifetime of a slice. We then elaborate on our proposals for intra-slice management, e.g., how to up/down-scale the resources of a slice, or migrate the VNFs of a slice chain, in a distributed manner to respect various types of SLAs, using data-driven mechanisms that use the AE and MS engines to adjust not only to the current slice and network status, but optimally consider future evolution as well. Finally, we describe our algorithmic solutions to cope with related reconfigurations and scaling events when multiple slices, possibly pertaining to different tenants, and (partially or fully) overlapping across domains.

2 Introduction and Problem Scope

In this section, we introduce first the concept of end-to-end slicing in 5G and beyond networks, and we motivate the need for the two main axes, along which MonB5G proposes to improve the state of the art in network slice management and orchestration.

The first axis is *a fully data-driven decision engine and algorithm stack.* While some recent 5G-related projects (e.g., PPP Phase 2) do investigate the introduction of AI- and ML-based solutions into the architecture, these either focus mostly on data analytics, and/or partial integration with the underlying orchestration architecture. In contrast, MonB5G investigates data-driven algorithms as the key ingredient of all slice lifecycle management operations, from admission control, to scaling and migration operations targeting a specific slice (we refer to these as *intra-slice* operation) and/or specific technological (e.g., RAN, MEC, core, cloud) and administrative domains, to end-to-end orchestration operations involving large number of slices each spanning multiple technological and possibly administrative domains. This is greatly facilitated by a tight integration with a highly sophisticated Analytic Engine (AE) and Monitoring System (MS), also heavily based on modern AI methods, that feed the algorithms of the Decision Engine (DE) with multi-grain, multi-modal, and tunable measurements and predictions, that are not available in existing architectures or related work. (The AE and MS systems are described in detail in deliverable D3.1 - here, we focus mainly on the interface(s) of the DE with these systems.)

The second key axis of novelty and performance improvement, is that the DE architecture is designed to be flexibly distributed across different administrative domains, technological domains, and even fine granularities such as one DE per tenant, slice, or even VNF (virtual network function). Flexible distribution means that the proposed DE could operate in different configurations, depending on the need. For example, a *hierarchical DE* implementation could be useful, where a "central" DE (e.g., running in the cloud or core domain) is in control of various "edge" DEs (partially) responsible for different RAN domains, edge clouds (e.g., MEC) etc., (or even slice-specific DEs, as mentioned earlier) all working towards optimal lifecycle management of slices spanning all these domains. There are great performance advantages of such DE/algorithm distribution:

(a) As the decision engine is envisioned to take actions at time granularities that are radically more short than traditional management operations (orders of minutes or hours), response *latency* becomes crucial; having the DE algorithms located as close as possible to the network components that the decisions are taken for, can greatly improve performance. For example, DE algorithms that orchestrate resources at a RAN domain often require decision delays in the order of milliseconds. The latency to collect information to and distribute actuation decisions from the central DE could be prohibitive; instead, such algorithms should be hosted at a local RAN DE, coordinating with the central one sparsely and on a per need basis.

(b) In addition to latency, the heavy use of data-driven methods for such algorithms suggests that the amount of information that needs to be collected and transported across the network is expected to be orders of magnitude more than in traditional monitoring and analytics for network management. It is well known that modern AI methods like (deep) reinforcement learning and deep neural networks, while often much more efficient than model-based algorithms in highly dynamic, complex environments, not only do

they require both great amounts of data during the training phase, but might also be quite burdensome during operation, if the trained model used is large (e.g., state-of-the-art deep neural network architectures used for image classification or other difficult ML tasks might have millions of trained weights – and thus resulting operations – that need to take place for an orchestration decision, e.g., the resource scaling for some local VNFs). Hence, in this deliverable, we will also present mechanism that not only distribute the conceptual architecture across domains (e.g., the same DNN model running with no, or minor modifications in different locations) but *distribute the model/algorithm itself across domains.* A key contribution of MonB5G in this direction is the concept of Distributed Deep Neural Networks, which can improve both decision latency and greatly reduce the network/communication footprint to achieve good performance.

Last but not least, in this deliverable we will also consider solutions that fully decentralize the DE, across domains, in a non-hierarchical, flat topology, without a "central" entity coordinating the rest. This setup is often referred to as *Federated Learning.* In beyond 5G networks, federated DEs are often applicable in scenarios where a slice spans multiple administrative domains, and the local DE of one domain needs to coordinate with other DEs to optimize a common goal (e.g., the end-to-end performance/SLA of a slice spanning these domains), yet it does not want to reveal any data to other DEs, besides what is absolutely necessary, for privacy reasons.

As a final note, it is important to stress that, while the above two architectural components, data-driven algorithms and distributed management, that characterize the MonB5G decision engine, have considerable standalone benefits and challenges, *it is their proposed co-design, advocated in MonB5G, that brings significant added value on top of each method and existing works, and enables a truly ``zero-touch'' management architecture*. It is the data intensive modern AI methods that stress the need for flexibly distributed DE solutions, and it is distributed, large-scale modern optimization methods that thrive in the thrive in the "Big Data" context and are able to deliver novel tradeoffs between learning/decision accuracy and network efficiency.

2.1 Structure

The main structure of this deliverable is summarized in the following table, linking each section with the corresponding task.

Table 1 Deliverable Structure

Section	Description	Task(s)	Starting Month
2	Dedicated to properly introducing the vision of MonB5G. Furthermore, its second goal is to familiarize the reader with the basic assumptions we make on the slice representation, its modeling, the KPIs we define for the decision engine performance.		

3	Presents the related work on Slicing control, and its intersection with AI-based techniques.		
4	Lays the foundations for the Decision Engine Architecture and describes in detail the inner workings of the several components of the DE.	T4.4	M13
5	Is the section that describes contributions related to the *Admission Control* of the Decision Engine. It presents Algorithms and policies, along with their results, as measured by the KPIs defined for the admission controller.	T4.1	M7
6	Here we include contributions that relate to the *intra-slice orchestration/control* dimension of the DE. Having received knowledge from the analytics engine, the slice takes data-driven local (per domain) or global (e2e) decisions for resource scaling and migration.	T4.2	M7
7	This section focuses on the problem of inter-slice orchestration, which is essentially hierarchically the highest orchestrator. After having collected intelligence and measurements for all slices, the goal of this orchestrator to reconfigure (either only in one technological domain or all at once) multiple slices and migrate them with an ultimate the respect of KPIs and SLAs.	T4.3	M7

2.2 The Need of Distributed Slice Management

Network slicing aims at providing customized networks instances tailored for 3rd-party vertical actors' service provisioning. Specific requests for network services and/or virtual networks are expected to be issued by business vertical actors to an orchestration platform and reach the MonB5G portal as depicted in Figure 1.

In this regard, the first step of the slice Life-Cycle-Management (LCM) involves the definition of a network slice template, also referred as blueprint, which resumes all the operational parameters, performance characteristics, and the set of network functions (e.g., PNFs or VNFs) required to build the service. For instance, with reference to Figure 1, examples of services include:

- **URLLC slice** would benefit from a dedicated server instance placed at the edge of the network, together with most of the core network functionalities. Enough transport capacity should be allocated to guarantee high communication standards, as those expected in Industry 4.0 scenarios, and radio interfaces should be tuned to guarantee adequate bandwidth.

- **eMBB slice** will require for a Cache VNF placed at the edge of the network in order to optimize the traffic distribution in the downlink direction, but most of the core network functionalities can still be located in the cloud premises due to less stringent latency requirements.
- **mMTC slice** will pose most of the effort in defining RAN specifications to guarantee inter-machine communication within pre-defined boundaries but will require limited cloud resources due to generally limited mobility in IoT scenarios.

The possibility to actually deploy the network slice generally depends on the resource availability and physical mobile network architecture. Therefore, the initial slice representation must be compared with the real network topology in order to define the initial VNF placement and chaining exploiting underlying NFV and SDN functionalities.

Due to its higher position in the reference architecture, this task is generally performed by the DMO. Its orchestration decisions rely on aggregated and real-time monitoring information which reduce the possibility of erroneous instantiation of the network slice services, e.g., due to node congestion. While a centralized decision approach would benefit from a holistic view of the network, it would provide poor scalability when dealing with realistic scenarios involving operational network topologies and introduce significant monitoring overhead.

Nevertheless, due to highly variable traffic loads as well as dynamic mobility patterns which affect the service demand, over time the initial placement strategy may result in sub-optimal settings negatively affect the overall operational conditions, e.g., making it impossible to further admit network slices.

In order to tackle these challenges, the MonB5G project design the DE as a logically distributed entity in charge of performing admission and control decisions as well resolving domain-specific resource allocation tasks limiting the involvement of the higher architectural layers. On the one side, this approach allows to quickly match the infrastructure resource allocation with the fast-changing underlying traffic demand, increasing the resource utilization and reducing operational costs. On the other side, it makes possible to instantiate domain-specific DE agents with limited scope and control-plane capabilities, reducing the amount of information exchange along the process and increasing the overall network efficiency.

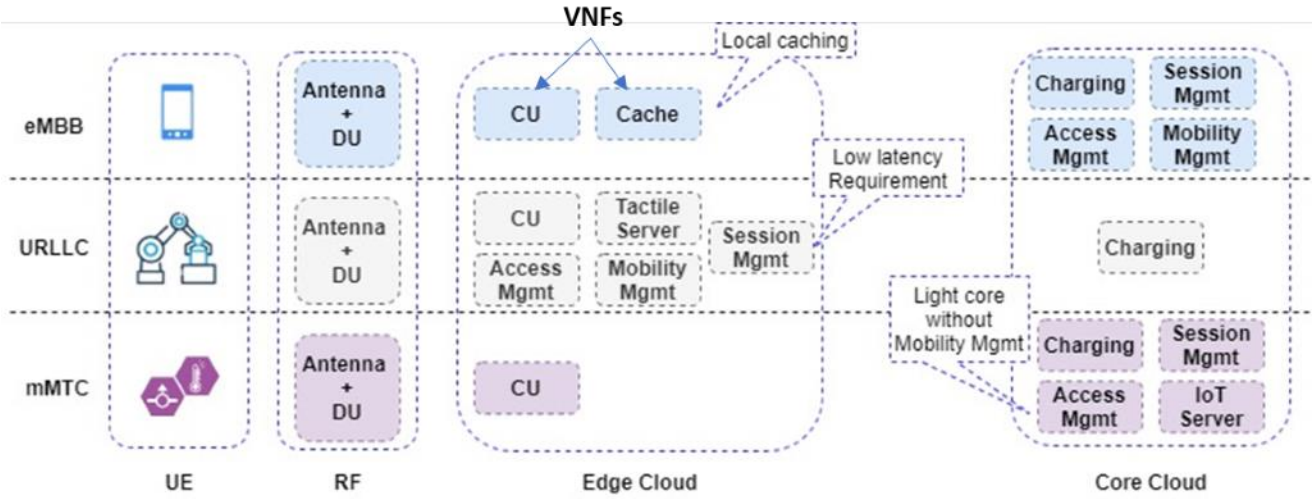


Figure 1 Initial slice's VNF placement according to the service graph [Modified and reproduced from 5G-Courses.com]

2.3 Slice representation

2.3.1 MATHEMATICAL DESCRIPTION

A core issue in network slicing includes describing a network slice requirement to define whether a slice request can be admitted in the system, and how to deal with it through its lifecycle management from a network operational perspective. The wide set of orchestration actions required to deal with the life-cycle management of network slices, inevitably brought the adoption of heterogeneous slice representations under practical circumstances.

The adoption of virtualization technologies in networking is driving a deep innovation in the way network services are deployed, managed and delivered. Thanks to the latest development in the field, traditionally hardware-based networking functionalities can now run as independent, and cost-efficient Virtual Network Functions (VNF) over a shared platform. This novel paradigm allows some network services to be provisioned in a more flexible way, therefore increasing the network capabilities when coping with unpredictable traffic demands.

Depending on the type of slice and its service requirements, several networking functionalities should logically interconnect, and be allowed to exchange information, i.e., a service function chain has to be created to sustain the service. In this context, an optimized deployment of network services, composed of VNFs that may be instantiated in multiple Data Centers (DCs) or Mobile Edge Computing (MEC) platforms, is one of the most challenging orchestration targets, as an accurate deployment of dedicated slice resources favours the availability of bandwidth and the satisfaction of latency requirements, especially in case of URLLC slice deployment.

The problem further exacerbates when Physical Network Functions (PNFs) are involved in the process, as vendor-specific management functionalities increase the need for dedicated interfaces to orchestrate the

resource sharing. The service function chain corresponding to a slice instantiation can be represented as a connected graph $G = (E, V)$ where each node (V) represents a VNF, and edges (E) are the links between nodes. Clearly, each edge should have enough capacity to handle the full traffic required by the corresponding service, and each node should have enough resources to handle the provided service.

Given the wide set of heterogeneous resources distributed over multiple technological domains, when dealing with resource allocation tasks, additional layers of abstraction as well as the definition of aggregated KPIs come into play, helping to achieve closed form mathematical formulations and maintain the problem tractable.

A slice can be defined as a subset of network resources allocated to an external tenant (virtual operator or service provider), with complete control over those resources. To this aim, the specific network domain plays a key role in the definition of required resources depending on the specific type of services. As an example, at the RAN domain spectrum-level slicing is considered, wherein the available radio resources (i.e., resource blocks) can be sliced by time, space, or frequency multiplexing, or by an overlaid access. Transport domains involves flow-based metrics, as well as topological characteristics such as number of nodes and edges composing the network, together with capacity and bandwidth features. At the core of the network, cloud-based metrics are often adopted, such as computing and storage capacity, to provide a representation of the computational requirements dictated by the service to be deployed.

Therefore, network slice requests are also represented as a tuple of different physical resources, e.g., radio RBs, CPU usage, storage, etc. considering the different technological domain of interest. This eases the admission and control task, where demand requirements are compared against infrastructure resource availability (defined by means of several abstractions) therefore allowing for the adoption of well-known optimization formulations, e.g., Knapsack, to solve the network slice allocation problem.

In this deliverable, we also assume that a slice *i,* and its placement over the mobile network infrastructure, can be represented by a graph $G_i = (V_i, E_i)$, where each node V_i represents a physical node of the network, i.e., a base station, a MEC platform, or a cloud server, candidate to host some slice functionalities, and each edge E_i represents a physical (or even logical) link that interconnects the nodes and allows for the creation of a specific network slice VNF chain. Additionally, each VNF is accompanied by a set of values; more specifically, the nodes in the RAN come with a load intensity which represents the resource blocks that are needed for the slice, the nodes at the MEC and at the cloud come with a requirement which stands for the computation resources, and finally, the links of the graph are characterized by traffic demand intensity.

What makes slicing hard, in general, is that the problem of deploying a slice G_i in a telecom network boils down to embedding smaller graphs (our slice tenants) to some larger graph (the infrastructure of the operator), $G_o = (V_o, E_o)$, where the subscript "o" stands for the telecom operator. We need to stress here that all nodes and links of G_o represent physical resources with some finite capacity, and that all the elements of G_i have some demand that needs to be appropriately (feasibly) placed in G_o. Nevertheless, network dynamics and slice requirements may change over time, for example as a result of end-user mobility, making the overall problem even more difficult to be solved in a cost-efficient way.

2.3.2 *SLICE REPRESENTATION IN THE STANDARDS*

When it comes to network slices, the high-level definition given by NGMN (Next Generation Mobile Networks) considers a sub-network instance as a shared entity between multiple slices [1]. It also considers that a

network slice instance may be fully or partly, logically and/or physically, isolated from another network slice instances. There have been many attempts to come up with a common slice data model as a standard. However, most of the contributions focus only on parts of the problem, mostly, how an end-to-end slice should be defined. For instance, 3GPP defines a network slice as a dedicated logical network, which provides specific network characteristics and capabilities on a shared infrastructure. 3GPP considers Network Slice Template (NST) to deploy network slices. NST describes the network slice, contains data such as slice configuration, network capabilities, and specific requirements of the tenants [2].

ETSI NFV considers an NFV network service as a resource-centric view of a network slice, this can be the case when NSI contains one or many VNFs. This makes the network slice model partially dependent on the network service Model [3]. There have been few attempts to disassociate the technology from the network slice model, for instance, this IETF draft is based on YANG module [4]. Nevertheless, this draft went obsolete and did not continue. The Reason might be the different vision IETF took compared to 3GPP and ETSI.

From this literature, we can classify the existing network slice models into three main classes: a) service-based; b) resource-based; and c) hybrid-based.

a) Service-based: network slices are represented as a set of network services. The model described on ETSI's report [3] is service-based. Examples of an EU project that adopt this model is 5GTANGO.
b) Resource-based: Network slices are viewed as a set of infrastructure resources with service elements deployed on them. The advantage of this model is scalability, by removing the service elements, the resources can be used to host anything. MATILDA, for example, follows a resource-based model. MATILDA by design instantiate and manage the application-aware network slices. The slice metamodel proposed by MATILDA is based on VIM-managed components (e.g., computing, storage, and network physical infrastructure) and their components [5].
c) Hybrid-based: Network slices combine between a set of network services and infrastructure resources; this model does consider a mapping functionality between the two. This model tries to take the most out of the two approaches. MonB5G, integrate ETSI NFV (service-based) and ETSI MEC (infrastructure-based) within the same slice management environment. Therefore, the MonB5G slice model can be seen a hybrid-based.

2.4 Inputs, and Actions of the Decision Engine

2.4.1 INPUT MEASUREMENTS

In this deliverable, the main concern is the decision engine (DE). Put simply, we are interested in the control of the slicing network, with an ultimate goal to achieve optimal (or near-optimal) performance in a variety of metrics and KPIs. Before discussing these goals (performance metrics, KPIs, etc.), it is of crucial importance to specify the set of measurements, which will define the state of the system. We assume that the DEs at:

- **Intra-slice** level (between the same slice) has clear view of the traffic in the previous time steps for this slice. This means that there are measurements for RBs, VMs workload intensity, as well as the accurate measurements for the communication requirements between VNFs that talk with each other. In principle, the slice DE scheduler should also have the ability to *observe* the system rewards/costs. That is, the DE should be updated through the monitoring system to keep track of

fluctuations in traffic demands, and also through the analytics engine for less obvious metrics, such as SLA violations, or in general for end-to-end metrics that the DE is trying to optimize/balance. Essentially, this second part of input measurements should serve as the *feedback of the DE.*

- **Inter-slice** level (among different slices), there exist the same measurements as above (traffic intensities for all active VNFs etc.), but equally importantly, the inter-slice DE *must have a holistic view* on the system configuration, i.e., which slices VNFs are placed in which infrastructure server nodes.

These inputs are supposed to be provided by the monitoring system (MS), and the analytics engine (AE).

2.4.2 *ACTION SPACE*

Conditioned on the gathered intelligence acquired by the analytics engine, and the monitoring system (the measurable inputs of the DE), the decision-maker (engine) should be at the position to take some action, towards improving the system KPIs. Although, these will be discussed in detail in Section 4, here we briefly list the control entities of the DE in an abstract way:

- *Resource Scaling*: When some slice observes its (past) traffic intensities from a set of base stations, it shapes a view about what its future might be. Therefore, it can generate a control signal that notifies the core network to release or reserve resources accordingly. This control action is closely related to the intra-slice level control of the system.
- *VNF Migration:* A really important control of the DE, mostly on interslice level, is to have the flexibility to migrate (move) the VNFs in different server nodes, with an ultimate goal to maximize some KPIs. Essentially, the DE should have the ability to define the configuration (embedding) of the slices at will.
- *Admission Policy:* A major control of the infrastructure operator is the policy which decides if new slices should (or could) be admitted. Loosely, this reduces to a “yes/no” action, whose consequences are however enormous. Admitting new slices could myopically result to increased revenue, but can end up in a skyrocket of delays, which result in SLA violations.

2.4.3 *SYSTEM PERFORMANCE METRICS*

There are many ways to define “how well” a decision engine performs. To this end, we make two distinctions. First, we focus on the actual performance of the decision engine (DE). In principle, a DE performs some actions in order to maximize (or minimize) some objective criterion that is of great importance to the system. Depending on the viewpoint of the designer, we can list a number of objectives equally important, and we initially discuss objectives that are network-operator related.

- *Minimum Server Usage*: Given the limited availability of physical resources, the DE should schedule the slice tenants VNFs placement to servers avoiding resource wastage. From the mobile network perspective, this is beneficial as it allows for:
    - Operational costs reduction,
    - Ease the admission of new slices, as “there is always room to fit new customers”, and

  - Energy efficiency, and thus greener communications.
- *Reconfiguration Operations*: The DE should have the necessary vision, which it should get from "learning statistical demand patterns", such that it will not move VNFs from server nodes unless this can lead to even better performance in the long run.
- *Service Level Agreement Violations*: A significant fraction of the costs is the amount that operators have to pay whenever they break their agreements with the slice tenants. If the operator has two VNFs of demands d_1 and d_2 (of slice 1 and 2), and the capacity of the node that it decides to place these two is $C < d_1 + d_2$, the operator will have to pay some amount to one of the two, or both, depending on its policies and fairness criteria.
- *End to End Delay across VNF Chain*: This metric should be measured across all technological domains. Essentially, a data packet that starts from MEC and has to reach the user, for some specific slice, has to pass through a set of server nodes (MEC->RAN->user). Depending on the DE placement, some slices may, or may not experience communication delays. If two slices need to share a server node, then it is probable that at some point in time, one (or possibly both) will have to queue. These queueing, plus the service delays, are added up and form the total delay of some slice.
- *Utilization of the resources:* A node that is not operating at full capacity still pays some cost to be ON (that is independent of the utilization). Depending of the scenario for example, VNFs supporting delay-tolerant services may be migrated to minimize the footprint of the VNF chain and maximize the resource utilization, ensuring that costs are amortized.
- *Slice admission rate:* In a sense, this and the above metric are closely related, but are not exactly the same. As an example, if the operator has 10 units of resources, and leases it to one tenant who pays 10 units, is the same as having two tenants who pay 7, and 3 units respectively (again for a total of 10), then high utilization suffices, and admission rate has no particular meaning. However, in most cases, there is a concave rule (principle of diminishing returns) according to which the operator leases, which implies that accepting more tenants will lead to higher revenue for the operator. In that case, the admission rate becomes of major importance.

2.4.4 *DISCUSSION ON THE PERFORMANCE METRICS.*

The relationship between the various objectives/metrics to be optimized is often non-trivial and conflicting. This requires sophisticated algorithms to find the right tradeoff, namely data-driven ones given the constant need to make predictions about the future state or impact of decisions. Minimum server usage suggests that the VNFs/processes will be packed, however, if there is a peak of traffic for even one slice, the performance of the whole system will experience a severe degradation, as sudden and high SLA violations will occur. Furthermore, having a very high admission rate, could mean an extremely bad end-to-end delay. Thus, if one for example, wishes to maximize the probability of slice admission, they should guarantee some other metric, and treat it as a constraint, because accepting everything would cause chaos to the performance.

A DE algorithm can try to tackle/tradeoff multiple objectives in different ways. One is to consider a multi-objective metric and attempt to find pareto-optimal tradeoffs. Another is to consider some of these metrics

as hard or soft (namely penalty functions) constraints. To summarize, we see that the choosing an objective, and constraints for the control problems is no easy task, and ultimately depends on the needs of the operators.

2.4.5 *MACHINE LEARNING ALGORITHMS KPIS.*

It is clear from the previous discussion that data-driven optimization algorithms will be involved both in terms of the key input parameter prediction (AE) as well as to identify good tradeoffs between the various conflicting DE goals. To this end, we also need a second set of KPIs that we'll use to gauge the performance of the ML algorithms themselves, beyond achieving (or not) good network-related KPI performance

These have to do with "how realistic is the DE", and if it is able to perform well in a real-world scenario. A non-exhaustive list follows:

- *Convergence (of Learning) Speed*: A fundamental problem of learning-based algorithms is that they typically need a lot of time to converge. Really slow learning can be detrimental, because as long as the agent has not learnt its surroundings, it is behaving sub-optimally. This relates to what ML community refers to as sample complexity, i.e., how many samples are needed to learn.
- *Scalability*: Another important issue we need to take into account, is how much the increase of the problem size affects the convergence speed. Ideally, one would want an algorithm whose convergence speed is as insensitive as possible to the size of the problem (states, actions).
- *Robustness to Dynamic Environment*: This is related to the "zero-touch" notion, which is an important aspect, and of the key strengths of MonB5G. Essentially, a learning agent can be tuned to understand its surroundings quickly, and then exploit its rewards, however, it is crucial that a data-driven algorithm cannot only learn an unknown environment, but that it learns environments that might be constantly changing.

2.5 Types of SLA

The Service Level Agreement is a contract between the Infrastructure provider and the Slice tenant defining the service characteristics and the responsibilities of each part. It states clearly the obligations of the provider in terms of QoS, hence it provides specific QoS parameters that must be satisfied. In case the QoS terms are violated, the provider is obliged to pay a penalty to the tenant. This penalty may take different forms (linear, nonlinear, etc.). The SLA depends strongly on the type of the slice since each type has different QoS requirements. Moreover, the SLA can be customized to the specific needs of the tenant. Some of the typical QoS parameters in SLAs are the following:

- *Minimum guaranteed bit rate/throughput.* It specifies the minimum value of successfully delivered bits per second for a particular service.
- *Maximum end to end latency.* It is the latency perceived by the end user. It includes both the delay due to propagation through the traversed links and the delay due to processing at the network's nodes [6], [7].

- *Service availability.* The provided service must be available within the agreed Quality of Service for at least a specified percentage of time [7].
- *Reliability.* Since a service is available, reliability is defined as the percentage of sent packets which are successfully delivered to the destination within the time constraint required by the specific service. Some of the factors that degrade reliability are hardware faults of the servers, software faults at different levels and operator's faults (e.g., mistaken network configuration or VNF deployment / migration) [6], [8].

SLAs may contain one or more of the above KPIs. It is true that usually it is nontrivial to map a high level QoS term to the actual physical resources that should be provided in order to satisfy the SLA.

A case where SLA violation is observed is when the reserved capacity provisioned in the network for a slice is lower than the actual slice demand. In that case the QoS is definitely degraded. The SLA can be formulated as a constraint of the slice embedding problem in following form:

- Hard (per time step) constraint: the provisioned capacity must be always greater than the demand
- Long-term constraint: the provisioned capacity must be greater than the demand X% of the time

3 Related work on AI for Beyond 5G orchestration

A significant challenge for 5G Networks is to support diverse verticals with very different Quality of Service (QoS) requirements. In addition, it is also expected that 5G Networks are able to support a large number of verticals simultaneously, with sometimes multiple services supported by each vertical. These requirements make the problem of resource management and orchestration one of the most critical in the context of 5G.

5G is designed to make extensive use of Network Slicing, which is one of its key technologies. Network Slicing enables the configuration of virtual networks by creating logical instances of a subset of the physical resources of the physical network. A varying amount and varying types of resources can be instantiated to create a network slice to provide different services with different performance constraints, making slices highly customizable. In this setting, the physical resources of the network are shared among the network slices, and these slices are owned by a tenant, which is the service provider or virtual network operator [9]. There can be a large number of network slices simultaneously deployed with varying degrees of physical resource sharing [10] [11] [12]. Thus, it is necessary to have orchestration mechanisms to coordinate the slices and to coordinate the resource allocation.

To meet the QoS requirements of the network slices and to ensure communication without downtime, researchers have focused on automating resource management with AI algorithms, since manual solutions for these problems are infeasible. The authors in [13] give a generic framework of the Network Slicing problem. A network slice consists of a set of interconnected Virtual Network Functions (VNFs) that make up a virtual network, which needs to be embedded on top of the physical network. This requires for VNFs to be placed on physical nodes. This is done by reserving adequate processing capacity, while the virtual links that connect VNFs must be associated with physical links by reserving sufficient communication capacity. The Network Slicing problem is a combined optimization problem that can be formulated as an extended Virtual Network Embedding (VNE) problem. This problem is known to be *NP-hard*. The goal is the optimal placement of a Slice (VNFs and associated virtual links) on the physical network, considering constraints on the capacities of the resources as well as on the placement's location (some VNFs run only on RAN nodes while others only on Core nodes).

Limitations of the State of the Art: The utilization of AI techniques to solve these kinds of problems is one of the main topics in the state-of-the-art research for 5G. There are still many challenges that lie in the efficient management, orchestration and configuration of VNFs and consequently the efficient allocation of resources. Despite the originality and the novelties of current state of the art, **there are still some important limitations**. More specifically, a large part of these works focus their entire contributions to the RAN domain, without providing an integrated e2e network slicing solution. The e2e solutions that were developed to overcome these limitations constitute a first step towards addressing the technical challenges behind the effective design of e2e slicing. However, advanced practical data-driven solutions are still required to handle the massive number of slices that are expected to operate in a recursive manner and in different timescales. Finally, the approaches that are tailored to specific applications do not provide holistic solutions for the instantiation (creation), and management of e2e network slices.

Beyond State of the Art: MonB5G will go beyond the existing state of the art network slicing solutions in several directions. In particular, we will design and implement different "layers" of network slicing, from coarse-grained high level slices that satisfy the operator SLAs to fine-grained slicing techniques that go even up to the user/application level. To that end, advanced ML concepts will be leveraged to process the big data volume in different parts of the network (e.g., MEC, RAN, core network), enabling an automated zero-touch virtual and physical function chaining that dynamically binds the heterogeneous resources (i.e., computational, storage, communication) in an end-to-end manner across different network domains. Finally, the proposed ML-aided network slicing mechanisms will be able to satisfy various distinct KPIs (e.g., delay, rate, QoE, massive connectivity), thus being application-independent and completely transparent to the end user, applicable to the whole cohort of 5G use cases.

3.1 Learning-assisted Admission Control

The problem of admission control involves the deployment of the network slices over the physical infrastructure, as well as the admission of the User Service Requests (USRs) into the slices after they have been deployed [14]. As more slices get deployed and as more USRs arrive, it is necessary to ensure resource utilization efficiency and revenue maximization for the 5G infrastructure provider.

The problem of resource allocation [15] is intrinsic to admission control, in which physical resources have to be dynamically re-assigned to a slice in order to guarantee the performance constraints of the services provided by the slice. When considering newly deployed slices, there's a VNF embedding and collocation problem [16] [17] [18] [19] that needs to be solved, which includes a resource allocation sub-problem. There are multiple Reinforcement Learning (RL)-based solutions for this [20].

Once the slices are operating, USRs start arriving [12] [21]. And depending on the available resources and the load generated by the USRs, these USRs will either get through to the network, will be deferred to be processed at a later time or rejected. The USRs arrive at the slices owned by the respective service provider at random times, and the rate of arrival is usually modelled as a Poisson distribution [22] (the same distribution is also used for slice deployment requests [23]). Since the slices present variable loads (i.e., different number of users with different loads as well), it will be necessary to re-allocate the available resources among VNFs executing in one single host, or to migrate the VNFs to different hosts without breaking the forwarding graph (i.e., the connectivity) of the Service Chains of the slice.

The problem of Virtual Network Function-Forwarding Graph (VNF-FG) embedding is targeted by Quang et. al. [24]. This problem relates to the initial deployment of network slices over the infrastructure. Quang et. al. proposed a solution based on enhanced versions of the Deep Deterministic Policy Gradient (DDPG), a continuous-action RL algorithm. Their solution neglects the cost of VNF migrations as a simplification, and focuses only on the impact of VNF collocation.

Bunyakitanon et. al. [25] also target the VNF placement problem and their solution relies on the forecasting of the Total Response Time for a VNF running an end-to-end service. The forecasting is done using multiple regression models (Lasso, KNN-Regression, SVR), as well as Decision Trees and Random Forests. This mechanism for VNF placement assists existing MANO frameworks into deciding the best placement for VNFs on the infrastructure hosts. However, it does not consider link/bandwidth resource utilization.

In [26], Bega et. al. implemented the Q-Learning algorithm (a form of RL) to make admission control decisions of the slices (not the USRs). Their approach works by modeling the admission control problem as a Semi-MDP in which a slice can be admitted or rejected (losing them forever) for deployment. The algorithm then learns to accept or reject slice admissions based on the maximization of the revenue perceived by the infrastructure provider after many admittance decisions have taken place, resulting in a better performance compared to other heuristics used as baseline comparisons. Their approach is very innovative, but does not offer comparisons with more careful defined heuristics for the problem domain or other RL approaches, which makes it difficult to assess the performance gains from using Q-Learning.

Han et. al. [23] used a genetic algorithm (comparable to RL algorithms) to determine an admission control policy from a set of admission strategies encoded as binary sequences. In this case, when the slices are not admitted, the response to the slice (admitted or denied) can be delayed instead of being denied directly, and their approach queues all initially declined slice requests. The slices have a utility value, and the purpose of the genetic algorithm is to maximize the overall utility value over a long term of T operation periods. One limitation in this work is the fact that their approach requires admission strategies already pre-defined and encoded, which may in fact not be optimal and non-trivial to determine. Moreover, encoding a policy in binary sequences is also not a trivial task depending on the specific policies, which adds an additional layer of complexity.

There have been multiple research efforts that seek to leverage AI mechanisms for admission control of the USRs. For example, the works presented by Buyakar et. al. [27], Sciancalepore et. al. [28], and Song et. al. [29] presented a traffic forecasting technique to control the admission of USRs. Salhab et. al. [30] also used a traffic forecasting approach, but used it to drive slice deployment requests and *slice scheduling*, rather than on USRs.

The framework developed in [29] evaluates the USRs in order to drive a resource orchestration mechanism, while in [27], the criterion for USR admittance is based on the USRs' SLA constraints in terms of bandwidth and delay. In the latter work, their approach developed uses an End-To-End delay predictor implemented using Mondrian Random Forests to predict the delay of an incoming flow. They also use a LSTM-based Machine Learning predictor to forecast bandwidth utilization. Based on these forecasts, their mechanism re-allocates resources to the slices in order to ensure that SLAs of the current and incoming USRs are met.

Sciancalepore et. al. [28] use the additive version of the Holt-Winters algorithm to forecast the traffic load of USRs, and extend their traffic predictor to capture the spatial distribution of the load in the geographical space in which the base stations are deployed. And to improve the capabilities of their traffic predictor, they implement a form of RL that monitors the occurrence of SLA violations of the slices. Using this monitoring mechanism, they change the parameters of the traffic predictor in order to reduce the *probability* of SLA violations. One limitation from their approach is the utilization of the Holt-Winters algorithm for prediction, since other machine learning-based predictors, like those based on LSTMs, can perform better in this respect.

Similarly, Salvat et. al. [21] also employed a machine learning approach to forecast traffic behaviors. In this case, they used the multiplicative version of the Holt-Winters algorithm to perform traffic prediction of the slices that are deployed. This prediction is then used to drive an admission control and resource reservation mechanism among the slices exploiting the fact that the users (USRs) rarely consume *all* the resources they request.

Limitations of the State of the Art: Using machine learning-based forecasting techniques has become one of the main approaches to anticipate resource demand of network slices, as we can see in the related work. These predictions can be exploited to design policies for admission control and resource allocation. The type of predictor used and the accuracy of the prediction are of extreme importance for these policies. However, a lot of the research works in the state of the art seek to improve on the accuracy of the prediction, which in itself cannot guarantee optimal admission control and resource management policies designed around it.

Beyond the State of the Art: MonB5G proposes to extend the state of the art in this respect, by developing more refined prediction mechanism that are trained using information relevant to beyond-5G technologies. Building on these novel prediction mechanisms, it is possible to enhance resource allocation and admission control policies optimizing them for different problem domains related to network slice management. Further enhancements to current state of the art will also include the design of distributed and decentralized forms of forecasting, which will bring as a consequence admission control and resource allocation mechanisms that are distributed as well.

3.2 Centralized Machine-Learning for Slice Management and Control

The slice embedding problem has been examined extensively by various authors and it would be useful to remark some of the different models and objectives used. In [31], the service chain embedding problem is tackled by maximizing the total flow. The authors propose an approximation algorithm that considers both the VNF placement and the routing of the flows. The setup is a network that comprises a number of computational nodes connected by links, without any distinction between RAN and Core Network (CN). The input is a number of different source-destination pairs of nodes, where each pair is associated with a maximum flow and a processing demand per unit traffic demand. The problem is to find the routing paths for all pairs that maximize the total flow by enforcing strict constraints concerning the bandwidth of each link, the processing power of each node and the maximum flow on each pair. This optimization problem is formulated as a mixed integer program which is *NP-hard*. The proposed method is an approximation algorithm that solves the problem in polynomial time combined with a heuristic algorithm that further improves the solution.

In [32], the problem of slice embedding is formulated using an objective that jointly takes into account the profit of the service provider and the profit of the network operator by considering the pricing of network resources. The chosen pricing policy seems to be important because it affects the profit of both parties and the demand for resources (which affects resource efficiency). The system is modeled by a weighted graph that comprises access nodes, forwarding nodes and data centers connected by links. Consequently, each VNF can be deployed only on a subset of the nodes. The total traffic demand for each slice is divided into different flows and each flow is associated with a source node and a destination node. The path of a flow must traverse the nodes where the slice's VNFs are deployed following a specified order. Iterative approximation methods are used to jointly optimize the slice embedding and the pricing of the resources.

In [7], the objective is to minimize the total deployment cost for the network operator, which includes the server cost, the cost associated with the VNF licenses and the link cost (it is related to bandwidth leasing). The authors model the system by a graph similarly to previously mentioned works. The interesting part is the hard constraints considered, which include not only the link bandwidths and the node capacities, but also the

minimum availability of service, the maximum end to end latency and the minimum data rate. This feature is important considering the diverse QoS requirements of Slices in 5G Networks. Note that availability may be reduced due to failure of the host server, the VNF software or the link. Moreover, latency is incurred due to processing or propagation delay and the data rate is limited due to the limited capability of a VNF instance to process traffic. The authors formulate this problem as an ILP and they also propose a more computationally efficient greedy heuristic algorithm.

There are also some works that examine the Service Chain routing problem in isolation from VNF placement. An example is [33], where authors model the network as an undirected graph with different types of nodes. The objective is to route an incoming service chain request so as to minimize the maximum network congestion. A service chain request is defined by a source node, a terminal node, the set of required VNFs and the traffic along the paths between consecutive VNFs, in which some these may have traffic changing effects (i.e., the amount of traffic coming out of a VNF might be different than the amount of traffic that goes in). This problem is formulated as an ILP problem as well, and computationally intractable to solve. Hence, the authors propose a routing algorithm inspired by the virtual circuit routing problem that is less computationally intensive and suitable for the online problem.

In [34], the authors present an overview of slicing in mobile networks that stresses the trade-offs introduced and the importance of dynamic resource allocation. High customization of services comes with a cost because it reduces the efficiency of resource sharing and impacts the operational costs. The most important contribution of the paper is a comparison between different slicing techniques that use static or dynamic allocation of resources to slices (dynamic allocation assumes that an oracle gives the exact future traffic demands). The metric used for their evaluation is the multiplexing efficiency (the ratio between resources required with network slicing and those needed under perfect sharing). The analysis is based on real traffic measurements, in a big and a medium city, acquired by a network operator. This analysis demonstrates the importance of dynamic resource allocation for increasing the efficiency of the network and the need for appropriate intelligent algorithms that will be able to work in short time scales.

Dynamic resource allocation requires real-time information about the traffic demand of each Slice. Moreover, an estimate of the future traffic demand based on the current demand would enhance resource allocation efficiency. Towards this direction, Deep Learning methods have been applied to facilitate resource allocation. In [35] the authors propose a method for approximating the traffic demands of individual services/slices by utilizing only the information of the total traffic volume monitoring. This problem is called Mobile Traffic Decomposition (MTD) problem. To accomplish this task, they introduce a novel framework that assists the employment of different neural network architectures to solve the MTD problem. Deep Learning methods can successfully estimate the individual service traffic demands by exploiting hidden spatiotemporal features (characteristic of each service) in the traffic aggregates. The proposed method is less computationally intensive compared to the in-depth inspection of the traffic streams and can be used to facilitate real time resource allocation.

In [36] the authors propose a Deep Neural Network Architecture that leverages the traffic measurement data of a slice at antenna level to make a capacity forecast for that slice. This optimizer predicts the capacity that should be reserved in order to reassure that no under-provisioning will occur and that over-provisioning will be minimized. The use of an appropriate loss function allows their approach to balance between under and

overprovisioning. This is one of the most important features of this model and makes it stand out from other forecasting solutions that perform prediction based on minimization of the error of the actual and future demands. Another important feature is its configurability and can be tuned for different reconfiguration periods according to the Network level.

Limitations of the State of the Art: The mechanisms proposed in the state of the art, as we have examined, solve different problems for slice management (dynamic resource allocation, slice deployment, forecasting, etc) and they rely on the measurement and pre-processing of different key performance metrics or indicators (KPIs). As the number of KPIs to be measured and pre-process increase, so does the complexity of the proposed mechanism and its computational load.

Beyond the State of the Art: In order to enhance further the capability of these mechanisms, solve these limitations and to extend the state of the art in this respect, **MonB5G proposes** the use of local Analytics Engine (AE) algorithms. These algorithms are expected to provide updated KPI estimates, complementing confidence and raw metrics, based on (Deep) Bayesian Networks and Gaussian processes. The latter is able to quantify the confidence in the ability of the locally optimized/reconfigured slice to resolve the potential problem (e.g., slice performance predicted to violate some SLA metric). Local and remote DEs will use AE-produced KPI estimates to reconfigure a specific slice based on (deep) reinforcement learning algorithms that will be responsible to both decide which features are relevant to be communicated, and which chain specific actions (scaling up/down, migration, etc.) to initiate. Finally, we will investigate multi-agent RL algorithms to implement a decomposition of the proposed DE algorithms.

In regards to the service chain routing problem and the slice embedding problem in particular, **MonB5G will extend the state of the art** by proposing appropriate graph structures and new deep learning methods optimized for them, that are able to accurately yet minimally capture inter-slice dependencies. These dependencies include (i) the coupling between potentially large numbers of resources, where re-configuring one slice can affect numerous others and their respective SLAs; (ii) the coupling between slices is partial, which creates complicated dependencies that give rise to hard optimization problems, beyond the sharing of individual resources between users or even slices. While deep-learning has had considerable success on image and sound-like inputs, graph-based relations have significant differences from the former, and existing methods are not directly applicable. To this end, MonB5G proposes as well to adapt novel methods from the recently emerging field of graph-specific deep learning methods to extract the key dependencies between slices. These will be combined with RL methods (local DEs) and deep RL (central DE) to efficiently (re-)allocate radio, transport, computation, and storage resources between massive numbers of co-existing slices.

3.3 Distributed Methods and Multi-agent Learning for Slice Management

Due to the enormous scale that 5G networks are expected to reach and the predicted future demand for even faster and more reliable communication services, researchers quickly shifted their focus to distributed methods and algorithms for slice management. Specifically, the problem of effectively deploying and managing network services in slices has raised considerable interest in the research community.

In recent bibliography, Shah et. al. [37] employ Multi-Agent Reinforcement Learning (MARL) to solve the Service Function Chaining (SFC) placement problem for Internet of Things (IoT) connected devices. The

presented system enables IoT devices to access processing power from the Network Function Virtualization (NFV) enabled network by sending requests and gaining access through SFCs that are deployed in the network. Their proposed solution is based on multiple Deep Q-Network (DQN) agents that map the SFCs to the substrate network and it is considered a resource allocation method.

In [38], Quang et. al., propose a multi-domain non cooperative VNF-FG embedding deep RL based approach. They consider a physical substrate network with multiple domains. One learning agent is assigned to each domain and it is trained to perform VNF-FG embedding in this domain using embedding cost minimization as its optimization criterion. To train their agents they adopt DDPG. Their proposal is innovative, since they are one of the first works to use multi-agent deep RL to solve the VNF-FG embedding problem considering non-cooperative domains. However, they do not consider the existence of multiple technological domains which limits the applicability of the proposed approach, and they neglect the cost of VNF migrations focusing only on the impact of VNF collocation.

In [39], the authors propose a Multi-Agent deep Q-learning method for federated and dynamic network slice management and resource allocation for differentiated QoS services in future IoT networks. Their approach considers optimization of resource allocation in terms of Transmission Power and Spreading Factor according to the slices QoS requirements. Simulation results show that the approach improves energy consumption, delay and throughput when compared with other traditional approaches. However, there is no real implementation of the proposed framework.

In [40], Sun et. al., focus on Radio Access Network Slicing. They propose a multi-agent RL-based Smart handoff policy with data sharing. The proposed policy aims to reduce handoff cost while maintaining user QoS requirements in RAN slicing. It consists in a distributed version of the Q-learning algorithm that is more efficient than centralized Q-learning due to smaller action space. Numerical results show that in the proposed distributed Q-learning approach can significantly reduce the handoff cost when compared with traditional handoff policies (without learning). Again, no real implementation of the proposed approach is provided.

One approach that has become a very important research topic is Federated Learning [41]. This is a recent Machine Learning paradigm that allows multiple agents to train a shared model in a decentralized manner without exchanging their local data. In particular, Federated Learning contrasts with "traditional" centralized Machine Learning model training in that it puts together elements from large-scale Machine Learning, privacy preservation and decentralized optimization [42]. Current research on Federated Learning is dedicated to address the many challenges that the approach presents, namely:

- Training a potentially huge model (neural networks can have millions of parameters) in a decentralized manner requires a large amount of communication. Current approaches to alleviate this drawback are trying to reduce the number of communication rounds .
- Coping with the large heterogeneity of the devices and communication channels in the network. The very different storage capacity, computing power and communication reliability of the devices connected to the network carrying out the require designing Federated Learning methods that are robust against lack or intermittent participation of certain agents. Current approaches to address this challenge are asynchronous communication, and accounting for the limited power of some components of the network instead of just ignoring them, which can have a negative impact on convergence [42].

- The data collected at each device may differ significantly in their distribution, which on the one hand violates the assumption of independent and identically distributed (IID) data, and on the other hand allows for the incorporation of other frameworks as multi-task learning [43].
- Achieving a good trade-off between model performance and quality of the exchanged information, since even just exchanging the model parameters and not the data itself can still reveal sensitive information [44].

Federated Learning techniques are particularly relevant to MonB5G because they deal with many implementation challenges of decentralized and distributed learning, which can be exploited to provide solutions of these same type for slice management. Similarly, approaches in the area of Decentralized Consensus Optimization (DCO) [45] can be used to train predictors, for example, using data locally collected at the analytic engines, dealing with decentralization and distribution learning similar to Federated Learning. DCO provides methods for coordinating the local predictions and take into account the data of all engines in the network. It can be seen as the theoretical optimization component of Federated Learning, since it tackles the problem of forcing the individual agents holding the data to agree on a unique common model that differs from that which each agent would have found in isolation, i.e., if no communication between the agents was possible.

Decentralized algorithms differ respect to distributed parallel algorithms (e.g. [46]) in that no central coordinator is required, therefore decentralized approaches are suitable for any network configuration. Furthermore, since there is no central coordinator, decentralized approaches must converge *by design* to a consensual solution (i.e., to the same parameter values) independently of the distribution of their data (i.e. they do not rely on the IID assumption).

Therefore, decentralized consensus algorithms (DCAs) are the methods to resort to for decentralized training of KPI predictors under the condition of keeping the data private at each node. Some DCA approaches have been developed to cope with the frequently changing networks. These are of particular interest in settings with detrimental connectivity conditions. Algorithms such as those found in [47] [48] for time-varying graphs can still guarantee convergence to the consensus solution under bounded delay. Alternatively, algorithms that are asynchronous and allow for one-to-one communication and customized network sampling [49] can potentially allow each node to adjust their communication schedule to minimize its idle time and that of its neighbors.

In the cross-domain case, some approaches focus on robustness against fluctuating communication speed and their potential interruption of the connectivity and network heterogeneity. Asynchronous decentralized algorithms have been proposed [48] [49] [50] that do not require all agents to perform their parameter updates simultaneously, and allow thus much more flexible implementation. Network load is another concern of the decentralized training, since a large amount of data must be exchanged through the network links. To alleviate the congestion, there are several decentralized compression algorithms that can achieve close-to-optimal performance with significantly less transmitted data [51] [52] [53].

Limitations of the State of the Art: The current state of the art works regarding multi-agent learning are focused on solving the SFC embedding and VNF placement problem approaching the solution in a non-scalable way. They are focused on traditional optimization algorithms, like Integer Linear Programming (ILP), to identify the placement that maximizes -or minimizes- a specific metric, or Single-Agent solutions, such as DQN, to learn an optimized placement for the entire network through trial and error. However, these

algorithms complexity and calculation time increases exponentially, posing problems with scaling in larger networks. Due to the predicted future demand, researchers shifted their focus to distributed methods and algorithms. In the recent bibliography, multi-agent Service Function Chaining (SFC) placement solutions have emerged, but mostly disrupting other fields and sectors, such as energy efficiency and security, rather than focusing on low latency-high bandwidth Ultra-reliable and low latency communications (URLLC) services.

Beyond the State of the Art: In the context of WP4 we go beyond the state of the art in distributed (federated) methods, along the following main directions: (a) we will investigate network-friendly federated and multi-agent learning methods that attempt to take the network conditions (e.g., congestions, link capacity, etc.) and/or network topology into account, and co-design the learning algorithm with the network one; (b) we will go beyond "static" state-of-the-art decentralized ("federated") optimization schemes towards online decentralized algorithms, based on the framework of Online Convex Optimization (OCO), which has been recently extended to distributed setups; (c) we will consider some recent advances on multi-agent reinforcement learning (RL) to extend the initially proposed RL- frameworks for end-to-end slice management, that are able to reduce the complexity of the state and action space by many orders of magnitude, with often limited optimality loss.

In regards to Multi-Agent Learning, MonB5G will extend the related State-of-The-Art by providing distributed Multi-Agent RL solutions to solve the Service Function Chaining (SFC) placement problem optimized for low latency, ensuring Quality of Service (QoS) requirements of critical URLLC services. Unlike similar solutions from the state of the art that utilize traditional optimization algorithms or single-agent RL, and only work in small scale system models, our solution will be able to optimize SFC placement keeping latency low, even in enormous sized networks with multiple domains.

3.4 Graph-based Learning for Slicing

The admission of slices at the infrastructure will trigger a series processes for VNF embedding and collocation [16] [17] [18] [19], which results in a resource allocation problem most of the time. This problem is called the VNF-Forwarding Graph problem [5], which consists on modeling the service chain functions of the slices as a forwarding graph.

Kuo et. al. [16] target the VNF-FG embedding problem by considering the problem as a *Joint VNF Placement and Path Selection* (JVPS) problem (which is NP-Hard), with the objective of maximizing the total size of admitted demands that can be actively processed. The model of the *physical* network they use is defined as an edge-weighted vertex-weighted directed graph G = (V, E), where the edges represent physical links and the nodes represent servers. In this model, the weight of each edge in E corresponds to the link capacities that it maps to and the weight of each vertex corresponds to the number of virtual machines (VMs) that it can host. The solution proposed, based on a dynamic programming algorithm, consists of analyzing the incoming demands for resources (not specified, could be USRs or slice deployment requests) separately and solving the JVPS problem given the current available resources for one at a time. Besides increasing the number of simultaneously served demands, their approach aims to increase resource efficiency by reusing VMs among multiple paths, as long as the constraints for each demand and path are satisfied.

Cao et. al. [54] provide a two-step solution for VNF-FG design that consists of flow designing and flow combining. In addition, they also provide a solution for placement of the VNFs into a VNF-FG and mapping them into physical nodes in order to minimize bandwidth consumption and maximize link utilization. The

formulation of this problem of multi-objective optimization problem (also known to be NP-Hard) is similar to Kuo et. al. [16]. However, Cao et. al. [54] do not show proof of its NP-hardness. Their solutions are four proposed genetic algorithms that are based on two well-known state of the art genetic algorithms, namely: multiple objective genetic algorithm (MOGA) and improved non-dominated sorting genetic algorithm (NSGA). They obtained their four algorithms by applying two different strategies, *random* and *greedy,* at the phase of initialization population. With the greedy strategy, nodes that have more residual resources will have higher priority during the VNF-to-node mapping, and links with shorter length are prioritized as well. Node mapping always occurs prior to link mapping. The four algorithms they used are: Greedy- MOGA, Random-MOGA, Random-NSGA-II and Greedy-NSGA-II, with the latter one performing the best.

The authors in [55] use a graph-based learning technique called Graph Convolutional Network (GCN) combined with deep learning and reinforcement learning to solve the Virtual Network Embedding (VNE) problem. As mentioned in the beginning of Section 3, the VNE problem can be considered a variant of the network slice placement problem. The authors proposed an Asynchronous Advantage Actor Critic (A3C) approach to solve it. They use a GCN layer in their model to automatically extract advanced spatial features of the physical substrate network to learn how to perform slice placement actions efficiently. Their work is innovative and shows better results than other recently published papers on deep RL for slice placement. However, their approach does not consider Quality of Service (QoS) requirements such as latency and it is centralized solution and no real implementation is provided.

The work in [56] introduces the idea of using a Digital Twin (DT) based on Graph Neural Networks (GNN) to capture the intertwined relationships among slices and monitor the end-to-end (E2E) metrics of slices deployed under a common infrastructure. The proposed DT exploits the novel Graph Neural Network model that can learn insights directly from slicing-enabled networks represented by non-Euclidean graph structures. Simulations are performed where the DT is used to mirror the network behavior and generate E2C metrics prediction of each slice. The model is show to predict E2E latency under different topologies accurately. However, the model is not tested here under a realistic environment.

Rkhami et. al. [57] deal also with a network slice placement and they combine DRL and Relational Graph Convolutional Networks (RGCN) in order to automatically learn how to improve the quality of an initial solution proposed by a heuristic. RGCN is introduced by [58] as an extension of regular GCNs to work with heterogeneous graphs. This propriety is essential to learn appropriate representations of large-scale relational data graphs. Simulation results show the effectiveness of the proposed approach but again the solution proposed is centralized and no real implementation is provided.

Limitations of the State of the Art: The use of graph-based ML through Reinforcement Learning or Deep Reinforcement Learning is adapted to the nature of network capabilities and features (nodes and links) and is very useful for automatic feature extraction to feed learning models for solutions based on Graph Convolutional Networks (GCN). However, the use of Reinforcement Learning or Deep Reinforcement Learning solutions alone **in current state of the art may lead to behaviors and decisions** that are not always explainable, easy to explore and sometimes not that efficient regarding the policies to apply (placement or resource allocation issues).

Beyond State of the Art: MonB5G proposes to extend the state of the art by developing techniques that couple heuristics with Deep Reinforcement Learning yielding more optimal control algorithms that can adapt their behavior after the learning phases.

4 MonB5G DE structure and specifications

The support for end-to-end network slicing provided by the MonB5G architecture passes through the definition of accurate data-driven decision schemes involving multiple network domains to guarantee per-slice Quality of Service (QoS) provisioning and the global KPIs already defined in MonB5G. Therefore, this section discusses DE specifications, functions, interfaces, and cross-domain operation.

The section first starts by positioning DE in the MonB5G architecture and shows that together with the other management elements, i.e., Monitoring System (MS) (including an internal memory), Analytic Engine (AE) as well as Actuators (ACT) form all a strong framework that **fit very well with the recently proposed ETSI ZSM Closed Loop (CL) automation architecture.** The section then delves into the definition and thorough description of the different DE interfaces, either inside a slice or with external entities in the MonB5G distributed system, and **provides afterward the detailed blocks of the DE in a generic way** that can be instantiated differently according to the specific needs of the target domain, slice, or network function, while linking it with the three main functions, i.e., **slice admission control, intra-slice reconfiguration and inter-slice reconfiguration.** This includes in particular the structure of the inputs (state space) and outputs (action space) as well as the design of reward/feedback signal. **The section then moves to show how MonB5G distributed architecture can be leveraged in decentralized decision either intra-domain (between different slices) or cross-domain, involving the end-to-end DE and several local DEs**. This is further clarified through practical Deep Reinforcement algorithms to show the nature of cooperation between the different DEs.

4.1 MonB5G DE vs. ETSI ZSM Control

In this section, we provide a mapping between the functional blocks of MonB5G control architecture that is built around the Decision Engine (DE) on one hand and ETSI ZSM Closed Loop (CL) automation framework on the other hand.

4.1.1 ZSM CLOSED LOOP CONTROL

ETSI has recently proposed some insights on the Zero-touch Service Management (ZSM) framework for Closed Loop (CL) management automation [59]. CLs may exist in each of the management domains of the ZSM architecture. Next Figure presents the CL functional scheme, in which the 'Decision' stage plays a key role. The aim of this section is to briefly present the ZSM framework proposed by ETSI and link it with MonB5G DE. As will be shown in the next subsections, the structure and specification of the MonB5G DE are aligned with it.

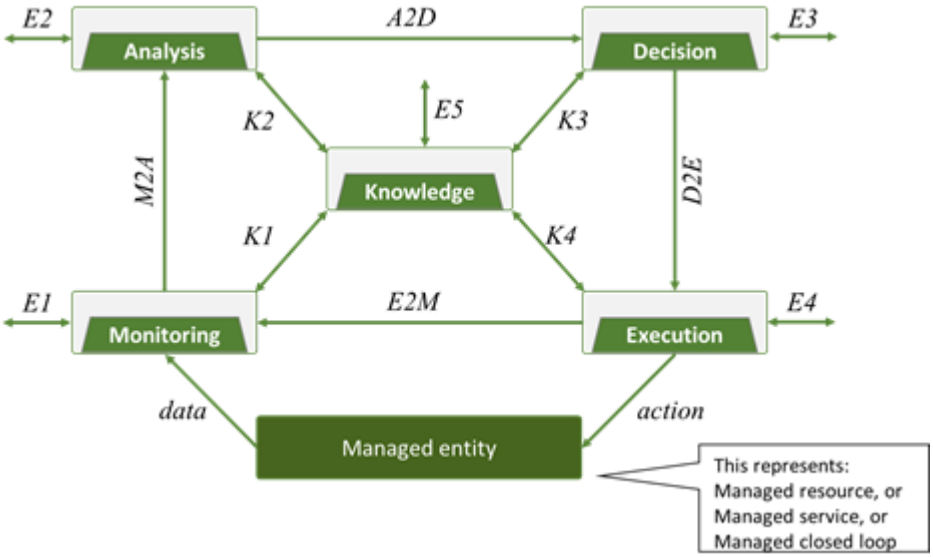


Figure 2. Functional view of a closed loop and its functional blocks within the ZSM framework [59]

This Figure is composed by four stages besides the 'Knowledge' functional block. The 'monitoring/collection' stage is the responsible for gathering and pre-processing the raw data from the managed entities or external resources (in this context, a managed entity is either a service, a managed resource, or another closed loop). Since raw data can have different heterogeneous formats, coming from different sources, can be transformed in a way that it allows to analyze it in conjunction with the data coming from other sources. After this, comes the 'Analysis' stage, which is in charge of providing insights from the available data obtained from the Monitoring stage. Then, the **'Decision' stage**, which governs managed entity, decides the action that must be taken based on the issues detected by the analysis block. These actions can be reactive, proactive, or predictive. It should be remarked that the decision stage is only responsible for deciding which actions are necessary, but not for their execution. This is the competence of the 'Execution' stage, which translates the decided actions into commands. The Execution stage is in charge of executing the necessary workflows in the managed entity in order to implement the actions determined by the decision stage. This execution could also involve other management domains. When this happens, interactions with other CLs are needed. Thus, multiple distributed CLs are required for the automation of E2E service management. The 'Knowledge' block in the functional diagram presented above is not technically a stage of the CL, it refers to the storage and retrieval of historical, configuration and operational data that are shared between the stages of a closed loop as well as between different CLs in the network.

The 'Decision' stage involves different primary flows or interfaces:

- **A2D interface**. It connects the Analysis function with the Decision stage. It provides insights on historical and/or real-time information provided by the collection/monitoring stage. It can also provide information for tuning the analytic models and starting/terminating the analytics processes.

- **D2E interface**. It is used by the Decision stage to provide action plans in form of workflows (e.g., configuration changes, onboarding services and resources). It can also be used to provide information to tune the decision models and start/stop the decision processes.
- **E3 external interface** represents the data and control inputs and outputs from/to other closed loops or external entities. It can be used to: i) start/stop the decision processes; ii) change the settings of the Decisions stage and attributes of the models; iii) retrieve the historical or real-time data of the function, such as logs, outcomes of the Decision function; iv) provide the resulting data of the Decision stage to other closed loops or authorized entities outside the ZSM framework, e.g., external management systems.
- **K3 interface** is a knowledge-enable flow, which is represented by a double-headed arrow, and is used for data-related inputs and outputs from the Decision stage. The primary interfaces exposed above can be augmented by data stored and retrieved from the 'Knowledge' functional block. The data can be historical workflows (generated over the time or coming from external resources) or real-time workflows (continuously generated by the operations of the Decision stage).

4.1.2 DE POSITIONING IN MONB5G ARCHITECTURE

In MonB5G it has been assumed that the **DE that is part of slice runtime management, is embedded in the slice** and is therefore part of the slice template. As depicted in Figure 3, MonB5G has separated in the template the slice management part, called Slice MonB5G Layer (SML) from the Slice Functional Layer (SFL).

The SML is split into sublayers responsible for Monitoring System (MS), Analytic Engines (AEs) and Decision Engines (DE) as well as the Actuators (ACTs). It is assumed that MS provides generic, reusable monitoring since it also contains **an internal memory that gathers measurements as well as decision history and AI performance metrics, serving thereby as a 'Knowledge block' comparable to ZSM proposal.** On the other hand, AEs perform data analysis and prediction. DEs implement control algorithms such as deep reinforcement learning to take online decisions regarding slice lifecycle management (LCM). Typically, in the management systems multiple goals have to be optimized therefore multiple AEs and DEs can be part of SML. To avoid competition between DEs, the DE Selector/Arbiter component is implemented (can be AI-driven). The SML enables direct, intent-based management by the tenant. **Therefore, MonB5G DE and its serving administrative elements (MS, AE, and ACT) are in line with the aforementioned ETSI ZSM CL framework.**

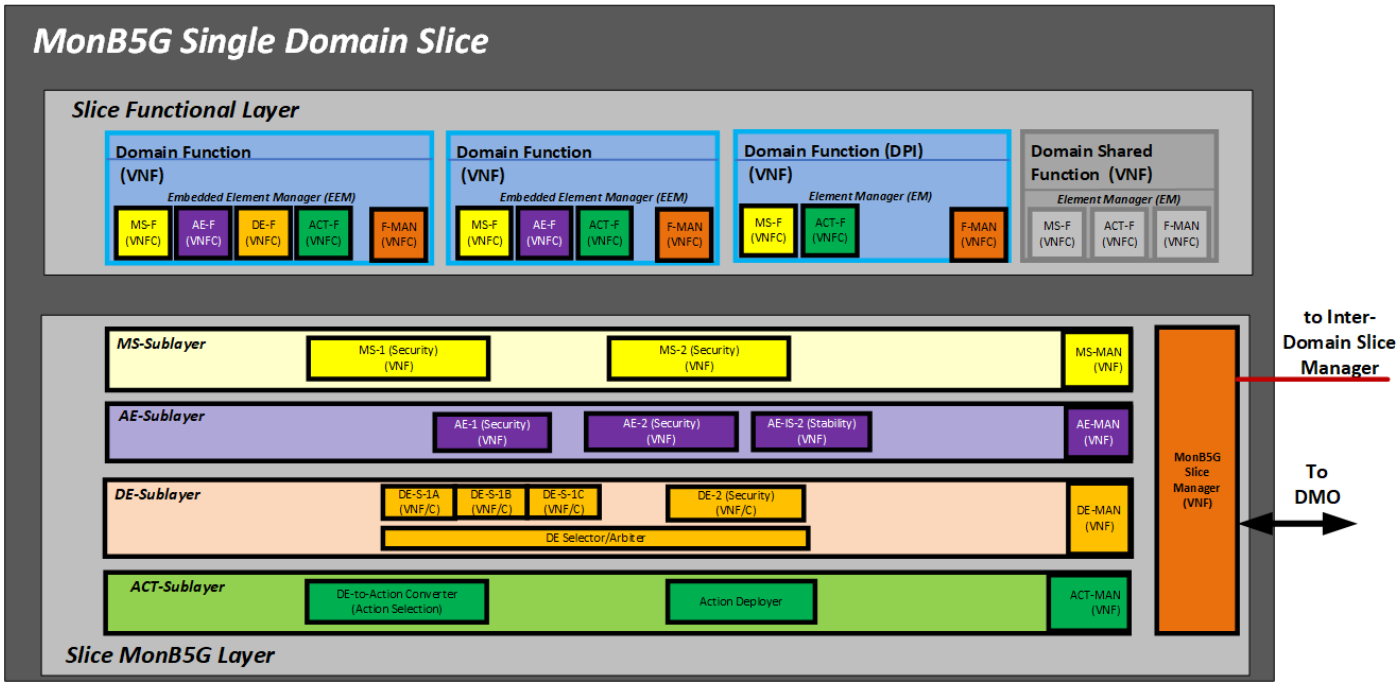


Figure 3 MonB5G DE Positioning with MS, AE and ACT

4.2 MonB5G DE Interfaces and Specifications

4.2.1 DE INTERFACES

Figure 4 shows how the DE, MS and AE are expected to communicate with each other. This figure also includes the actuators which translate DEs decisions into API calls to different slice components (e.g., VNFs, links, PNFs) in each of the technological domains (RAN, Edge, Cloud) that a slice is supposed to cross.

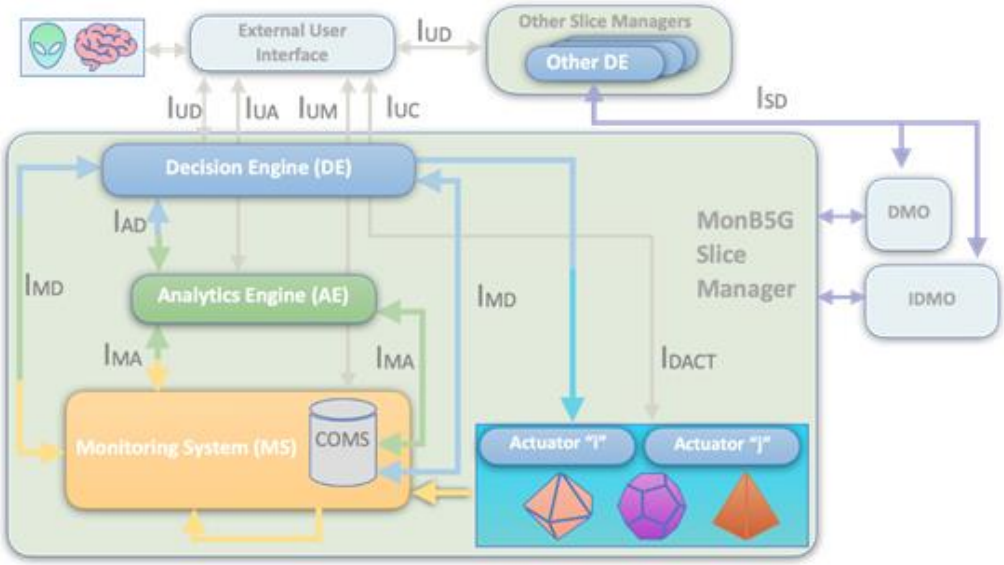


Figure 4 DE Interfaces

The MS in Figure 4 is the one responsible of gathering a set of different metrics from the systems that the DE is controlling. This information can be passed to the DE and the AE directly, but they are also stored in a common online memory store (COMS), illustrated by the grey cylinder. This COMS is added in order to avoid implementing hard synchronization constraints among the MS, DE, AE whenever information needs to be exchanged. In this way, the DE and AE can be more flexible in terms of the length of their processing without compromising the granularity at which the MS can sample monitoring data from the controlled systems. So, it is the MS (depending on its capabilities and amount of information as well as the granularity set from the **External User Interface (EUI)**) that somehow defines how fast the data is sampled**. It is worth emphasizing that COMS is in line with the 'Knowledge' block of the ETSI ZSM functional scheme presented in** Figure 2.

The AE then reads the monitoring information from the COMS in order to pre-process it (e.g., perform predictions) before making it available to the DE. The AE may also read information directly from the MS, but this is expected to be used in more punctual cases, where some synchronization is needed. **The prediction interval can also be set from the EUI**. Once the AE outputs the pre-processed data, it will proceed to store it in the COMS.

In a similar way, the DE is expected to read its input from the COMS, however it is also possible to receive this information directly from the AE and MS. Once the DE has generated its decisions, it will also store those in the COMS, as well as issue them to the actuator interfaces of the systems it is controlling to translate them into API calls for slice components lifecycle management (LCM). **DE parameters can be fine-tuned in runtime from EUI as well and might take effect in the next DE configuration update interval.**

Cross-domain operation between local DEs (i.e., DEs of each technological domain) or with E2E DE passes via the Inter-Domain Manger and Orchestrator (IDMO), while operation between inter-slice DEs inside is ensured by DMO.

Table 2 provides a description of the different interfaces that link MonB5G DE with the other control blocks.

Table 2 Description of the type of DE interfaces and the associated role

Interface	Type	Role
I_{AD}	Tensors/Database query	DE Reads the predicted KPI from AE (either online or from COMS)
I_{MD}	Tensors/Database query	DE reads raw MS measurements (either online or from COMS)/Store AI metrics and DE decisions in COMS
I_{MA}	Tensors/Database query	AE reads raw MS measurements/Store AI metrics, predictions in COMS
I_{UD}	Database Query	EUI reads/Changes DE configuration (e.g., discount factor of a DRL algorithm)
I_{UA}	Database Query	EUI reads/Changes AE configuration (e.g., prediction interval, learning rate)
I_{UM}	Database Query	EUI reads/changes MS configuration (e.g., granularity)
I_{UC}	Database Query	EUI reads/changes actuation configuration (e.g., API primitives' parameters)
I_{DACT}	REST API Call	DE sends decisions to Actuators

4.2.2 MONB5G DE SPECIFICATIONS

As we can see in Figure 5, the DE is composed of several internal sub-blocks. The Input/output Pre-processors, the Control Trigger (CT), and the Decision Algorithm (DA). The Input/output Pre-processors are customizable according to the adopted Decision Algorithm (RL algorithm, GA algorithm or something as simple as a PID). For instance, in the case of a deep reinforcement learning (DRL) decision, these pre-processors **correspond to the environment (e.g., OpenAI Gym-based) that converts the inputs received from either the MS or the AE to a standardized State Space representation and translates the actions of the Decision Algorithm to an Action Space format.** The decision algorithm might encompass several internal sub-blocks, such as replay buffers, actor, and critic neural networks in the case of DRL. The CT is added to make sure the DA executes when systems states are read and verified that they are consistent. This could be done on demand by an external user, or it could be done **periodically driven by the reward function of a DRL algorithm.**

Figure 5 Inner workings at the DE

MonB5G decision algorithm performs slice-level admission control, intra-slice reconfiguration (e.g., VNF scaling, VNF (re)-placement) as well as inter-slice reconfiguration. This requires access to a set of parameters in the state space to provide then the adequate action space instance based on a feedback/reward. To that end, we describe in the sequel a typical representation for both spaces as well as the key idea of the reward/feedback.

4.2.2.1 DE STATE SPACE

The state space of Domain DE includes but is not limited to the following set of measurable/predicted parameters:

- Initial map of new slice service graph into the architecture, i.e., which slice's VNFs should be created in the local domain (e.g., eMBB slice requires the creation of CU and Popular Content Cache VNFs at the Edge).
- Initial required resources per VNF for the new slice in the current domain. For the RAN, initial RAN resources (i.e., the PRBs).
- Edge/Cloud AE prediction of resource usage per VNF for the existing slices over the next T time steps. As an example of expected resources, we can list the following: CPU usage, RAM, disk space power available, the latency and the link bandwidth.
- RAN AE prediction of resource usage per slice, i.e., the UL/DL Resource Block Groups.
- RAN UL/DL scheduler type per slice. The scheduler involves several relevant functions, such as for instance, link adaptation, resource assignment, power control, etc. and different types of schedulers are assigned to satisfy the QoS requirements of the applications.

The end-to-end DE state space contains but is not limited to the following parameters:

- Initial required resources per edge/cloud VNF for the new slice (1 slice might have several VNFs per domain) and the initial RAN resources (PRBs) for the new slice.
- Domains where the target slice VNFs creation is successful and domain/s where the target slice VNFs creation failed (if any).
- Latency prediction and observed latency of all the slices. Since latency is an important KPI in some applications (e.g., URLLC scenarios), the latency must be properly measured and forecasted in order to fulfill the SLAs.
- Information of the free resources that are available in the other domains (e.g, CPU, RAM, disk space, etc.). This information is needed for future slice reconfigurations.

4.2.2.2 DE ACTION SPACE

The set of possible actions taken by DRL and transmitted to actuators to enforce them into the architecture are the next:

- **Slice admission control.** This action includes the initial VNF placement within the domain servers and the initial resource allocation. It should be noted that the former is a discrete action, while the latter is bounded continuous action.
- **Reconfigure a slice without changing other slices.** After the admission, change the VNF placement. If the domain AE predicts a resource usage decrease over now+T time steps, then scale down the corresponding resources (e.g., CPU, RAM, Disk space, Power, Latency, Link bandwidth, Resource Blocks). On the contrary, if AE predictions show that an SLA is about to be broken, then decide to change the VNF placement.
- **Inter-slice reconfiguration.** If Failed initial VNF placement/allocation or running VNF scale up: Change the placement of other slices' VNFs inside the domain to accommodate the new slice (statistical multiplexing) or scale the running slice. If AE predictions show a future resource usage decrease over now + T time steps, decide to scale down and give the resources to the demanding slice.
- **If a domain DE, e.g., Edge DE fails,** the E2E DE can intervene to perform a global reconfiguration. This includes moving an existing slice VNF from the Edge to the Cloud, if, for instance, the E2E AE predicts a relaxation in its latency requirement, which allows to allocate the released resources at the Edge to the existing or new slices.

4.2.2.3 REWARD/FEEDBACK

The reward function is carefully defined to guide the DA (generally a DRL agent) towards maximizing the slice admission rate and the resource utilization as well as optimal VNF placement/slices reconfiguration, while minimizing the total network cost (i.e., CPU consumption, latency, energy consumption and SLA violations). Therefore, the design of the reward function requires considering a multi-objective approach. This will be defined as a weighted linear combination of the costs to be maximized (e.g., the number of success admissions) and the inverse of those to be minimized (e.g., 1/energy, 1/number of reconfigurations, 1/SLA violations). In this multi-objective function, the weights of the costs need to be properly balanced and the constraints on the KPIs are enforced via a penalty in the reward. By tweaking the weights, the agent is guided to optimize the costs prioritized by the slice tenant and/or the infrastructure operator and that are set via the External User Interface (EUI).

4.3 DE Cross-Domain Operation

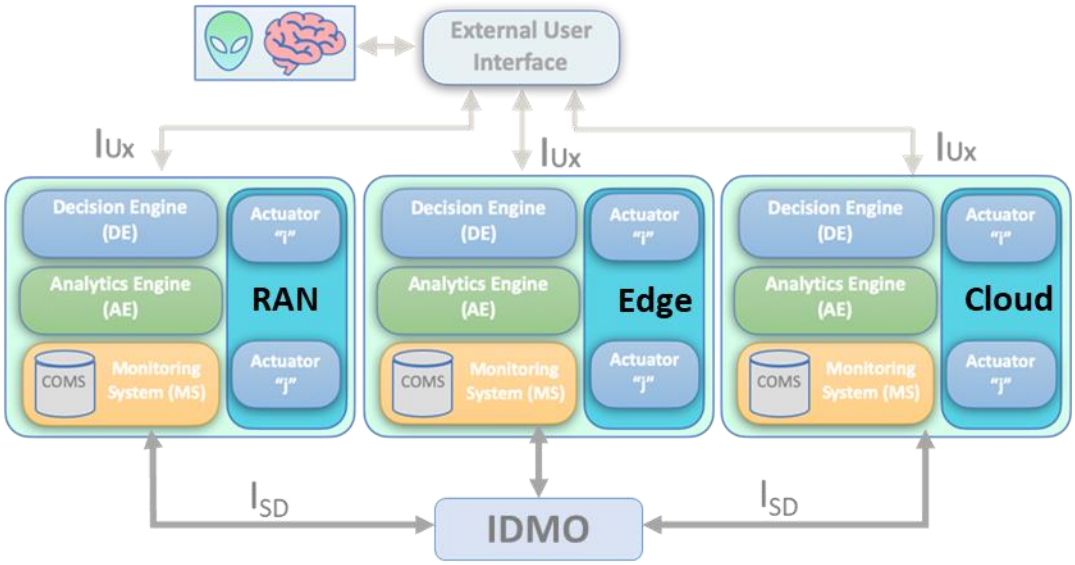


Figure 6 Multiple instances of the MS/AE/DE triplet across multiple technological domains

At every technological domain, it is expected that there is a MS/AE/DE triplet, as shown in Figure 6. However, we might have more instances of the triplet depending on the granularity of control we might need for the overall system, which could be per slice, per tenant, per domain, per Infrastructure Provider, or other. Given that there might be multiple instances of DE, the design of the DE and its interfaces has been made agnostic to the specifics of the technological domain or the system layer in which it is deployed, especially that DEs need to collaborate/cooperate to achieve the end-to-end control/decision that targets the optimisation of a key performance indicator (KPI).

Leveraging MonB5G distributed architecture with decentralized decision can be made via the use of several advanced combinations of decision schemes. To exemplify, we present here two approaches:

- **Federated Deep Reinforcement Learning.** In this scheme, the local DEs are built upon either value-based DRL (e.g., DQN) or policy-based DRL (e.g., A2C, DDPG, SAC). The key idea is that, as long as the action/state spaces of the different DEs are homogeneous, one can dramatically improve the decision accuracy and leverage the experiences learned in e.g., other domains, slices, by letting the local DEs exchange the weights of their embedded decision actor/critic or Q networks with the E2E DE that performs a custom averaging to generate a more accurate model and broadcasts it to the local DEs to update their local networks and improve decision. This avoids exchanging raw data between local and E2E domain, even in the decision phase.

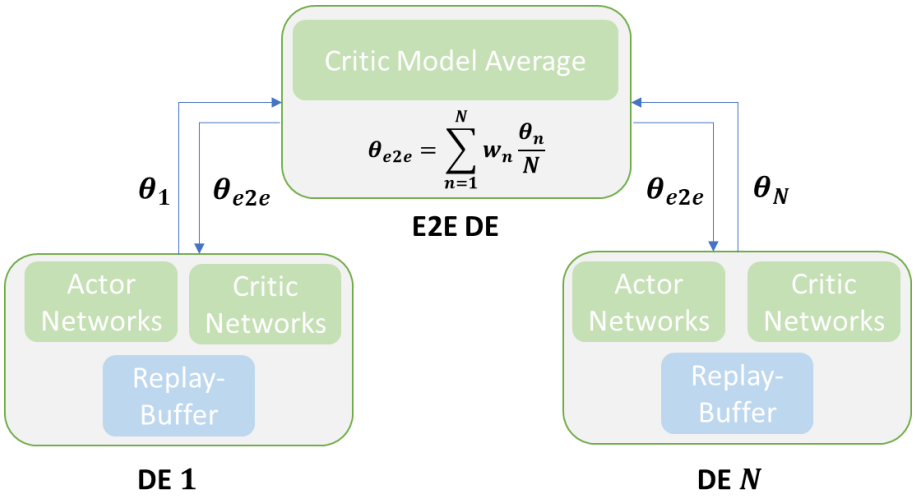


Figure 7 Federated DRL Leveraging Distributed Cross-Domain/Slice DE

- **Multi-Agent Deep Reinforcement Learning.** Local DEs play the role of Actors and can learn policies using only local information (i.e., their own observations in local domain, slice, etc) at execution. On the other hand, the Critic is located in the E2E domain. This allows the policies to use extra information to ease training, so long as this information is not used at test time. It is unnatural to do this with Q-learning, as the Q function generally cannot contain different information at training and test time. Thus, actor-critic policy gradient methods where the critic is augmented with extra information about the policies of other agents is suitable to implement a decentralized decision leveraging MonB5G architecture.

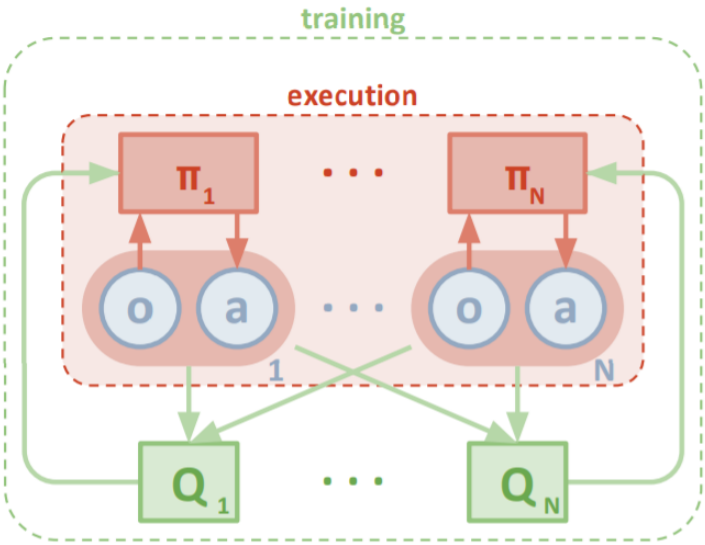


Figure 8 Multi-agent decentralized actor, centralized critic approach [60].

5 Slice Admission Control

A slice can be defined as a subset of network resources allocated to an external tenant (virtual operator or service provider), with complete control over those resources. The allocated resource are needed for the deployment of virtual functions and connectivity between them and slice customers. The resources are typically virtual (computing, storage, connectivity); however, some physical resources as PNFs or hardware accelerators can also be allocated to slices. The resources are typically distributed spatially (as customers also are), and the way in which they are allocated to a slice can impact seriously slice performance. The slice functions that are required to exchange information with low delay should be, for example, placed in the same datacentre, and the delay may also concern interactions between the slice and the users, according to a predefined metric such as delay (min, average or max) in some cases (for example for URLLC slices). The amount allocated to slice resources can be dynamic; however, a critical point is the initial allocation of resources to a slice that is used for slice admission. In some case, resources and their features can be included in the slice deployment request. In some cases, they can be estimated; in the other ones, the number of needed resources can be unknown. One of the goals of the MonB5G project is to make an initial resource allocation to a slice in such a way that would minimize the operations related to the change of resource allocation, i.e., the scaling of resources. The ultimate goal is to satisfy as many slice deployments requests as possible but at the same time fulfilling slice KPIs. Such approach maximizes the revenue of infrastructure providers.

In MonB5G, it is proposed to elaborate the decision about slice admission based on the following factors:

- In the case of multi-domain slices of the same type, different ways of the split of a slice into multiple domains with evaluation of the possibility of their mutual interconnection. Then the split slice graph should be deployed in which of involved domains as described below. In the process of splitting slice template, the IDMO is involved. The problem of admission of end-to-end slices deployed in multiple technological domains will be described in section 5.2.
- The first criterion that has been evaluated during slice admission process is the status of consumed and available resources and termination time of running slices (some of them can be short-lived slices). The status of resources should also include their reliability. In MonB5G the status of consumed resources will be combined on the predication of their usage in the future based on AI algorithms.
- The number of resources needed by a slice (min, max, average, initial) can be obtained:
    - Directly from a slice template.
    - Indirectly from a slice template using GST/NEST approach as proposed by GSMA [61]. In this case, each slice instantiation request has, for example, additional information about the number of users, etc. This information has to be during slice admission converted into resource demands.
    - Time of day when a slice is admitted and its duration. For most slice templates, the time-of-day may have an important impact on the number of active users, therefore on resource consumption.
    - The history of the resource consumption by the slice template. The history may show the min, max and average resources consumption and the dependency on the time of day. Such

historical measurements allow for the estimation of the number of resources needed by a slice with the minimum number of customers.

- Slice priority is yet another factor that has to be taken into account. Slice instances that handle real-time traffic need higher resource allocation priority than a slice without real-time constraints. Moreover, some service types may include resource allocation priorities such as e.g., in case of an emergency slices, which may require very high resource allocation priority. In general, resource allocation priority should be established on a per slice-basis. Priority per slice function or link is not considered in MonB5G; however, in some situations, such an approach might prove to be beneficial.
- Slice lifetime has to be taken into account as short-lived slices may be deployed in the case when some slices are already scheduled but no in the near future.
- Resource allocation constraints may concern reliability, delay, separation (impact of congestion in other slices) and some preferences related to the deployment of a group of functions (VNFs) in the same data centre, selection of the preferred data centre (close to users).
- Resource pricing combined with slice calendaring can be used for overall efficient resource consumption. If a slice can be deployed with a strong time of deployment constraints, its deployment can be shifted to a moment of time when the averaged resource consumption is relatively low (for example, deep at night). Such slice can be used, for example, for synchronization of databases, creating billing reports, etc.).
- The ability of a slice to work in an 'emergency mode'. Such emergency mode is a situation in which a deployed slice can minimize its resource consumption, but it is still able to provide basic functionality. The emergency mode can be used during disasters in which the number of resources is significantly reduced, but the slice functionality is essential for the control of the power grid, etc. Another case is a failure of the power grid system that may require switching off of some resources in order to keep the lifetime of the temporarily battery-powered data centres. So far, such a 'dual template' slice issue is so far ignored.
- RAN resource allocation is a separate problem as it may deal with a way in which the RAN slicing is implemented.

In general, the rule of keeping running the existing slices templates should be kept (golden rule in telecommunications). The exception are slices of higher priorities which deployment may cause termination of slices of lower priorities. An open issue is a negotiation between slice requester and slicing system owner that can lead to the deployment of slice instance using reduced resources in comparison to the initial request, instead of rejecting the request.

There are some cases in which slice deployment time is critical. Part of the time is an algorithmic delay and the time needed to access information that is needed for the elaboration of the decision. Therefore, in such cases, the admission control algorithm should be able to provide the admission decision quickly.

As it has been presented, the process of slice admission is multidimensional, and in such case, the AI approach seems to be the most efficient one. It has to be noted, however, that most of the present approaches use much simpler problem definition than described above. Typically, the problem of slice admission control is defined as the optimization of revenues of infrastructure operators (or maximization of the usage of infrastructure resources) while keeping the required KPIs of deployed slices (slice KPI/QoS violation can be linked with a penalty) [10].

The lifetime of a slice may vary, but we can assume that short-lived slices may stay for 30 mins and that can be long-lived slices that may stay for years (the MVNO case). It is, therefore, hard to estimate the number of slice requests in a specified period of time. In general, the slice request arrivals are non-deterministic, and in most cases, (except slice calendaring) average waiting time must be short enough. Typically slice admission strategies include random, first-come-first-served, priority-based, greedy and semi-greedy [10].

Network slice can span multiple and different technologically domains. In case of the 5G network, the domains comprise RAN, transport, MEC and Core. The resource allocation in each of the domains can be different – in RAN; for example, it is about Resource Blocks scheduling. In MonB5G multiple orchestrators and domain-specific templates are used. The admission of the end-to-end slice has to be preceded by the admission process in each of the domain that is used for the end-to end slice deployment. If admission is possible in each domain, the and-to end-slice is admitted for deployment. In MonB5G the multi-domain slice deployment requests are coordinated by IDMO, whereas, for each domain, a respective DMO for slice admission is used.

The forthcoming sections are dedicated to the slice admission control algorithms that can be used for domain-specific (section 5.1) and cross-domain admission control (5.2). These section presents the initial works of the MonB5G project.

5.1 Single domain slice admission control algorithms

As described already, the admission control for multi-domain slices can be in most cases decomposed into a set of independent admission control process in each domain that is used by the multi-domain slices. Therefore, the presented in this section domain-specific admission control is also applicable in case of slices spanning multiple technological or administrative domains. The efficient slice admission control impacts the 3.2 (maximization of network slice acceptance ratio) and 5.4 (reduced slice reconfigurations) KPIs of Deliverable D2.2.

5.1.1 CLOUD INFRASTRUCTURE BASED SLICE ADMISSION APPROACHES

This subsection presents a description of novel slice admission control (SAC) algorithms that can be run at the Infrastructure Provider (InfProv) level (cloud domain), in order to derive an optimal policy regarding acceptance or rejection of the arriving network slice request. The proposed algorithms are based on Reinforcement Learning, and seek the optimal policy to maximize InfProv revenue while reducing the penalty caused by SLA violations. Three algorithms that use the following techniques are introduced:

- Q-Learning (QL),
- Deep Q-Learning (DQL),
- Regret Matching (RM).

Besides deriving the optimal policy, the ability to run offline or online, which is a crucial criterion, is assessed for each method. The proposed system model is illustrated in Figure 10.

The model considers two main players: (i) the Infrastructure InfProv, which is the owner of the network infrastructure being in charge of instantiating network slices for tenants by providing the required resources;

(ii) the tenants that request the instantiation of network slice from the infrastructure provider to offer services for their clients.

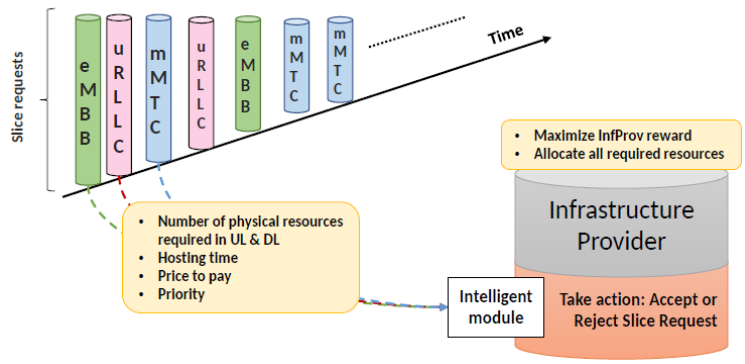


Figure 9 System model

Each network slice is characterized by five main criteria:

- **The physical resources** needed to satisfy the requirement of a network slice, in UL and DL, noted $Nres_{req}^{i}$(UL) and $Nres_{req}^{i}$(DL), respectively. Where *i* is the slice type, which can be either eMBB, uRLLC, or mMTC. Needed resources by each slice type are described as follows:

$$Nres_{req}^{eMBB}(DL) >> Nres_{req}^{eMBB}\ (\text{UL}) \quad (5.1)$$

$$Nres_{req}^{uRLLC}(\text{DL}) >> Nres_{req}^{uRLLC}\ (\text{UL}) \text{ or} \quad (5.2)$$

$$Nres_{req}^{uRLLC}(\text{DL}) == Nres_{req}^{uRLLC}\ (\text{UL}) \text{ or} \quad (5.3)$$

$$Nres_{req}^{uRLLC}(\text{DL}) << Nres_{req}^{uRLLC}(UL) \quad (5.4)$$

$$Nres_{req}^{mMTC}(\text{DL}) << Nres_{req}^{mMTC}\ (\text{UL}) \quad (5.5)$$

  We justify these assumptions as most eMBB traffic has dominated DL traffic (ex. high-definition video streaming), while mMTC traffic has dominated UL traffic (ex. IoT traffic). The case of uRLLC is different, as all the types of traffic may exist and depend on the service.

- **The hosting time of each network slice type H_{time}.** Each slice, if admitted, will use the InfProv's resources for a given duration.
- **The priority of the slice** depending on the application running on the corresponding slice.
- **The price P_{req}^{i}** that a slice tenant pays to InfProv for the used resources. The tenant has to pay the resources per time unit for the H_{time} duration ("req" refers to a slice tenant, and "i" refers to the slice type).

The infrastructure provider entity is characterized by its capacity in terms of available resources at time t (C_t). It represents the total amount of available resources that may be allocated to a new network slice. It is

worth noting that C_t is updated when a network slice is admitted or leaves. In other words, when a new network slice is accepted, the needed resources will be allocated and dedicated to it; when a network slice is terminated, its associated resources will be automatically released. Thus, at each time instant *t*, the available resources at InfProv is performed as follows:

$$C_t = C_{total} - C_{allocated} + C_{released} \quad (5.6)$$

Note that, two formulas are used, one for UL and one for DL, as the resources are separated. C_{total} is the total number of resources available in the InfProv in UL or DL. $C_{allocated}$ and $C_{released}$ are respectively the number of allocated and released resources at time t in UL or DL. The proposed SAC model is applied only for the RAN resources, composed of DL and UL. We argue this assumption by the fact that RAN is considered as the bottleneck of the system, while other network slice's required resources (such as computing) are always available, and no reservation is needed. In summary, the InfProv is characterized by its resources capacity C_{total}, while a network slice is identified by: $Nres^i_{req,}$ H_{time}, $Priority$, and P^i_{req}.

As stated earlier, we seek an optimal admission control policy that aims at finding a trade-off between fulfilling the network slice resource request (UL and DL); while maximizing InfProv revenues. First, we propose to model the SAC using Markov Decision Process (MDP) [62]. Since exactly solving the MDP is very challenging due to the difficulties in modeling the traffic dynamics, we apply reinforcement learning to derive the optimal policy and to find the earlier-mentioned trade-off. For that, we will use different Reinforcement learning models, namely QL, DQL, and RM, to predict the optimal action to apply when a new demand of a network slice arrives at the system (i.e., accept or reject an arrival slice request).

A Markov Decision Process is composed of 4-tuples *M = (S; A; T; R),* where *S* is the set of states, *A* is the set of actions, *T* is the transition probability from state *s* at time *t* to state *s*' at time *t + 1* when taking an action *a*, and *R* is the reward obtained by performing the action *a*, which leads to move from the state *s* to *s'*.

For our system, we assume that a state *s = (n, m, l, b)* is composed of four information where:

- *n* is the number of accepted eMBB slices;
- *m* is the number of accepted uRLLC slices;
- *l* is the number of accepted mMTC slices;
- *b* is a value that can be equal to 1, 2 or 3 to indicate the slice type, eMBB, uRLLC, or mMTC, respectively, of the last received request.

At receiving a new network slice request, InfProv, observes the state of the system via an agent and takes action a:

$$a = \begin{cases} 1 & if \text{ new arrival slice request is accepted} \\ 0 & if \text{ new arrival slice request is rejected} \end{cases} \quad (5.7)$$

The different transitions of the system occur when a new network slice arrives, and a decision is needed when a network slice leaves the system. If the system is in a state *s = (n, m, l, b)* and a new slice arrives, a decision needs to be taken (i.e., accept or reject), leading the system to transit to one of the following states:

- *(n + 1, m, l, 1*) if a slice of eMBB is accepted;
- *(n, m + 1, l, 2)* if a slice of uRLLC is accepted;

- *(n, m, l + 1, 3)* if a slice of mMTC is accepted;
- *(n, m, l, 1)* if a slice of eMBB is rejected;
- *(n, m, l, 2)* if a slice of uRLLC is rejected;
- *(n, m, l, 3)* if a slice of mMTC is rejected.

If a network slice leaves, then the system moves to one of the following states without taking any action:

- *(n − 1, m, l, 1)* if a slice of eMBB has left;
- *(n, m − 1, l, 2)* if a slice of uRLLC has left;
- *(n, m, l − 1, 3)* if a slice of mMTC has left.

As mentioned above, each network slice is described by <$Nres_{req}^{i}(DL), Nres_{req}^{i}(UL)$, H_{time}, $Priority$, and P_{req}^{i}>.

We assume that the price P_{req}^{i} to pay by each slice tenant, by time unit, is proportional to the slice priority. Hence, we propose to model the estimated reward that InfProv expects to receive from each accepted network slice as follows:

$$R_{inf} = \left[sign\left(C_t - Nres_{req}^{i}\right)\right] * P_{req}^{i} * Htime_{req}^{i} \qquad (5.8)$$

With:

$$sign\left(C_t - Nres_{req}^{i}\right) = \begin{cases} 1 & C_t \geq Nres_{req}^{i} \\ -1 & C_t < Nres_{req}^{i} \end{cases} \qquad (5.9)$$

In equation 5.8, we multiply the P_{req}^{i} by H_{time}, since each accepted slice tenant pays a P_{req}^{i} according to the slice priority by a time unit. Hence, the total price that a tenant of an accepted slice will pay depends on his priority and the requested hosting time. Besides, we have added the sign in this equation to ensure that there are enough resources in InfProv to support the number of required slice resources.

It is worth noting that in this work, for each accepted network slice, the InfProv should be able to provide the needed resources for both UL and DL. Otherwise, the InfProv will pay a penalty, if it accepts a slice request without having enough resources to cover the slice resource requirements. To this end, Ct should be always higher than $Nres_{req}^{i}$ in UL and DL. Therefore, the reward defined in 2 for both UL and DL will be calculated as follows:

$$R_{inf} = \left[sign\left(C_{InfDL} - Nres_{reqDL}^{i}\right) \wedge sign\left(C_{InfUL} - Nres_{reqUL}^{i}\right)\right] * P_{req}^{i} * Htime_{req}^{i} \qquad (5.10)$$

We also note that the R_{inf} is null if the slice request is rejected, as neither penalty nor reward can be applied. Having defined the MDP model, we need to find the optimal policy that maximizes the long-term total reward for InfProv. The optimal policy corresponds to the action to take for each state *s* aiming at maximizing the long-term total reward. Since the MDP is hard to solve using techniques like Value iteration or Policy iteration as the traffic model is hard to model, we describe in the next section how to find this policy using Reinforcement Learning, through three models QL, DQN, and RM.

5.1.1.1 ADMISSION CONTROL USING Q-LEARNING

Q-learning is an offline reinforcement learning algorithm that generates an optimal policy to maximize the expected total reward for any finite MDP [63], i.e., the state and action spaces may be finite, which is our model's case. This policy is based on the Q-learning function, which is designed to seek the best action in each state to maximize the long-term total reward. The QL method consists first of calculating, for each possible action in each state, a value named Q-value. Then the QL method stores these Q-values in a table, namely the Q-table. This step is called the exploration of the unknown environment. It is worth noting that the Q-table is initiated to zero and updated with the new Q-values obtained after each episode. The agent performs in a state s_t, one of the two actions: accept or reject a new slice request for the epoch t, and it observes the state transitions st+1, and rewards r. Hence, it updates the Q-value using the weighted average of the previous and the new Q-value, as shown in the following equation:

$$Q_{new}(s_t, a_t) \leftarrow (1-\alpha)\, Q(s_t, a_t) + \alpha\, (r_t + \gamma\, max_{\alpha \in A}\, Q(s_{t+1}, a)) \qquad (5.11)$$

With:

- $Q(s_t, a_t)$ is the old Q value;
- $Q_{ner}(s_t, a_t)$ the new value obtained after updating the old one;
- α is the learning rate that controls how fast the new estimation adapts to the random changes imposed by the environment.
- is the discount factor that notifies the importance of future rewards;
- r_t is a reward received from action a_t;
- $maxQ(s_{t+1}; a)$ is the estimation of the optimal future action.

After several episodes, the Q-table converges and becomes the reference table for the agent (.i.e., the entity that takes decisions) to select the best action based on the Q-value. However, one of the QL method's weaknesses is the convergence time, i.e., the time needed by the agent to explore all the states to learn the best action to take in the future. Indeed, it depends on the state space; if the latter is large, the time to converge is high, which may be problematic if QL is used without offline training.

5.1.1.2 ADMISSION CONTROL USING DQL

Q learning is based on a Q-table to store the learned results for each state and action. Consequently, if the state space is large, the table size explodes, leading to an increase in the training time as the agent has to take more time to explore all the states. To this end, DQL uses deep learning to represent the Q-values, where each state passes through several hidden layers of a neural network to get the Q-values [64]. Then, DQL calculates: (i) the loss function that represents the mean squared error (MSE)as shown equation 5.12 of the predicted Q-value (Q_{pred}), and (ii) the target Q-value (Q_{target}) (see equation 5.13) that represents the maximum possible value for the next state.

$$MSE(\theta_i) = [Q_{target} - Q(s_t, a, \theta_i]^2 \qquad (5.12)$$

$$Q_{target} = E[r + \gamma \max Q(s_t, a', \theta_{i-1}] \qquad (5.13)$$

With θ is the weight.

Using the same θ weights in (5.12), the values Q_{target} and Q_{pred} move at the same time. For this purpose, DQL uses two neuralnetworks, one for Q_{pred} and the other one for Q_{target}. Algorithm 1 presents the different steps of the DQL algorithm. Note that we have considered the states and actions as defined in the MDP.

Algorithm 1 Deep Q Learning algorithm

Ensure: $s \leftarrow s_{t+1}$
Initialize: replay memory D to capacity N
Initialize: $Q_0(s, a)$ with random weights,

for $episode \leftarrow 1$: M **do**
  Initialize state s_t

  for $t \leftarrow 1$: T **do**

- With probability ϵ select:
  - a random action a_t
  - $a_t = max_a$ Q(s_t, a, θ) (otherwise)
- Execute action a_t and observe reward r_t and state s_{t+1}
- Store transition (s_t, a_t, r_t, s_{t+1}) in D
- Set $s_{t+1} = s_t$
- Sample random minibatch of transitions (s_t, a_t, r_t, s_{t+1}) from D
- Set $Q_{target\,j} = \begin{cases} r_j \text{ for terminal } s_{t+1} \\ r_j + \gamma * max_{a'} * Q(s_{t+1}, a', \theta_{i-1}) \text{ for non terminal } s_{t+1} \end{cases}$
- Perform a gradient descent step:
- $MSE(\theta_i) = [Q_{target} - Q(s_t, a_j, \theta_i]^2$

  end for
end for

5.1.1.3 ADMISSION CONTROL USING REGRET MATCHING

Regret Matching (RM) is an online learning algorithm similar to Reinforcement Learning. Its agent (player or user) looks for the right action based on the regrets of the previous actions. The main principle consists of minimizing the regrets of its decisions at each step of the system [65]. To do so, the agent relies on past behavior of taken actions to guide its future decisions by favoring the actions that it regrets not to have chosen before. The strategy of this method is to adjust the agent's policy by distributing probabilities on all actions proportionally to the regrets of not having played other actions. The regret is defined as follows: if a is the action chosen by the agent at time T, thus for any other action $a \neq a^*$, the regret of choosing the action a but not another action a^* up to time T is obtained as shown equation 5.14 [66].

$$R^T(a, a^*) = \frac{1}{T}\sum_{t=0}^{T}(\, r_t(a)\,) - \frac{1}{T}\sum_{t=0}^{T}(\, r_t^i(a^*)\,) \qquad (5.14)$$

With: $r_t(a)$ is the reward obtained at time *T*, by choosing the action a. At each step, the agent chooses an action a_T between two actions (accept or reject) considered in this study (see equation 1). The probability P_{T+1} that the agent will choose action *a* in the next step defined by next time *T+1*, is defined as follows:

$$P_{T+1}(a)= \begin{cases} \frac{[Reg_T(a,a^*)]^+}{\sum_{a=0,1}[Reg_T(a,a^*)]^+} & if\ a \neq a_T \\ 1 - \sum_{a' \neq a_T} P_{T+1}(a') & if\ a = a_T \end{cases} \quad (5.15)$$

With: $[Reg_T(a,\ a^*)]^+ = \max[Reg_T(a,\ a^*), 0]$ presents the non-negative part of the regret $Reg_T(a,\ a^*)$. The RM algorithm is illustrated in Algorithm 2.

Algorithm 2 Regret matching algorithm

Ensure: $s \leftarrow s_{t+1}$

Initialize: action a_1,

for $t \leftarrow 1{:}\ T$ **do**

- Take action a_t
- Receive reward $r_t(a)$ and $r_t(a^*)$
- Update the regret of choosing action a_t and not a_t^* is: $Reg_T(a,a^*)= \frac{1}{T}\sum_{t=1}^{T} r_t(a) - \frac{1}{T}\sum_{t=1}^{T} r_t^i(a^*)$
- The probability of choosing action a_t in the next step is: $P_{T+1}(a)= \begin{cases} \frac{[Reg_T(a,a^*)]^+}{\sum_{a=0,1}[Reg_T(a,a^*)]^+} & \text{if a} \neq \text{a}_T \\ 1 - \sum_{a' \neq a_T} P_{T+1}(a') & \text{if a}= a_T \end{cases}$
- Select action corresponding to maximum probability
- Update action a_{t+1}

end for

It is worth noting that the RM is a fully online solution; the policy should be initiated before that the algorithm starts adapting itself. Therefore, we consider RM with two initial policies: accept and reject. The first one starts by accepting the network slice requests, while the second one starts by rejecting the requests.

5.1.1.4 PERFORMANCE EVALUATION

In this section, we present the simulation results of the slice admission control problem by comparing the three methods' performances. It is worth noting that the RM considered here is initialized once by accepting the first received requests (noted as RM with accept policy), and once by rejecting the first received requests (noted as RM with reject policy).

Table 3 Number of resources: (i) available in InfProv, (ii) requested by each slice in UL and DL

[UL, DL] InfProv	[100, 100]
[uRLLC, mMTC, mMBB] UL slices	[5,9,5]
[uRLLC, mMTC, mMBB] DL slices	[5,5,10]

We assume that InfProv receives requests to create slices following a Poisson process with two different arrival rates as follows: (i) rate=2 per time unit (tu) corresponding to a low arrival rate (i.e., the slice requests arrive rarely), and (ii) rate=10 per tu corresponding to a frequent slice request arrival. Besides, we assume that each slice request stays hosted in InfProv for a H_{time} period. To show the impact of H_{time}, we use four values as follows: (i) short period where H_{time}= 5 *tu*, (ii) medium periodwhere H_{time} = 20 *tu*, (iii) large period where H_{time} = 50 *tu*, and (vi) very large period where H_{time} = 100 *tu*. We consider that thenumber of resources requested by each slice in UL and DL, and the number of resources available in the InfProv are different, and their values are presented in Table 3. Note that the three algorithms considered in our work (i.e., RM, QL, and DQL) apply the same reward formula, which is based on the price and hosting time of each accepted slice request.

Regarding slices' priority, we assume obviously that uRLLC slices have the highest priority since it hosts application requiringcritical latency and reliability, while eMBB and mMTC slices have the same priority. The price of running the uRLLC slice type is four times higher than the price to pay for running the mMTC and eMBB slice types. The latter slices (i.e., mMMTC and eMBB) have the same price. In other words, we use the price to pay as a way to enforce priority among the slice types.

Considering the training session, it is worth noting that the offline version of both QL and DQL algorithms requires a training phase, in which the agent explores the environment to learn how to achieve the optimal and most rewarding actions. However,our results are generated on the test phase, i.e., we aim to evaluate our learning models on new data.

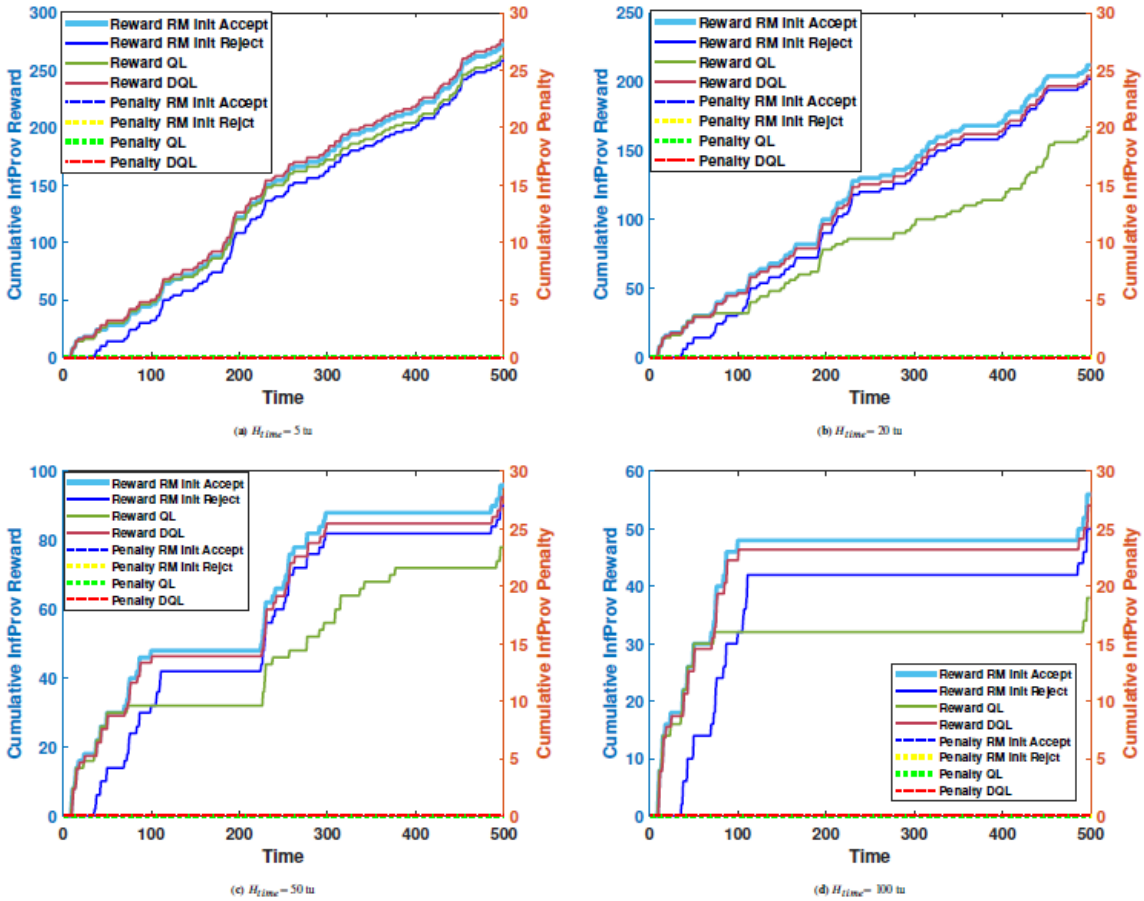


Figure 10 InfProv reward and penalty vs. Time for slice arrival request rate= 2

Figure 10 illustrates the cumulative reward as well as the cumulative penalty obtained using the proposed algorithms (RM initialized with accept policy, RM initialized with reject policy, QL, and DQL) when the arrival rate is 2 per *tu*, and for four values of the Holding time (H_{time}). The same metrics are shown in Figure 11, but for an arrival rate of 10 per *tu*. We recall that cumulative reward is obtained by the infrastructure provider when accepting slices, and (ii) the penalty is incurred when a slice is accepted but InfProv has not sufficient resources either in DL or UL or in both directions to satisfy its requirements. For the cumulative reward, we notice that, for both arrival rates and in the four proposed solutions, when the H_{time} increases, the cumulative reward decreases. We argue this by the fact that the resources are not released quickly when H_{time} is high; hence the InfProv rejects new requests during this H_{time} period.

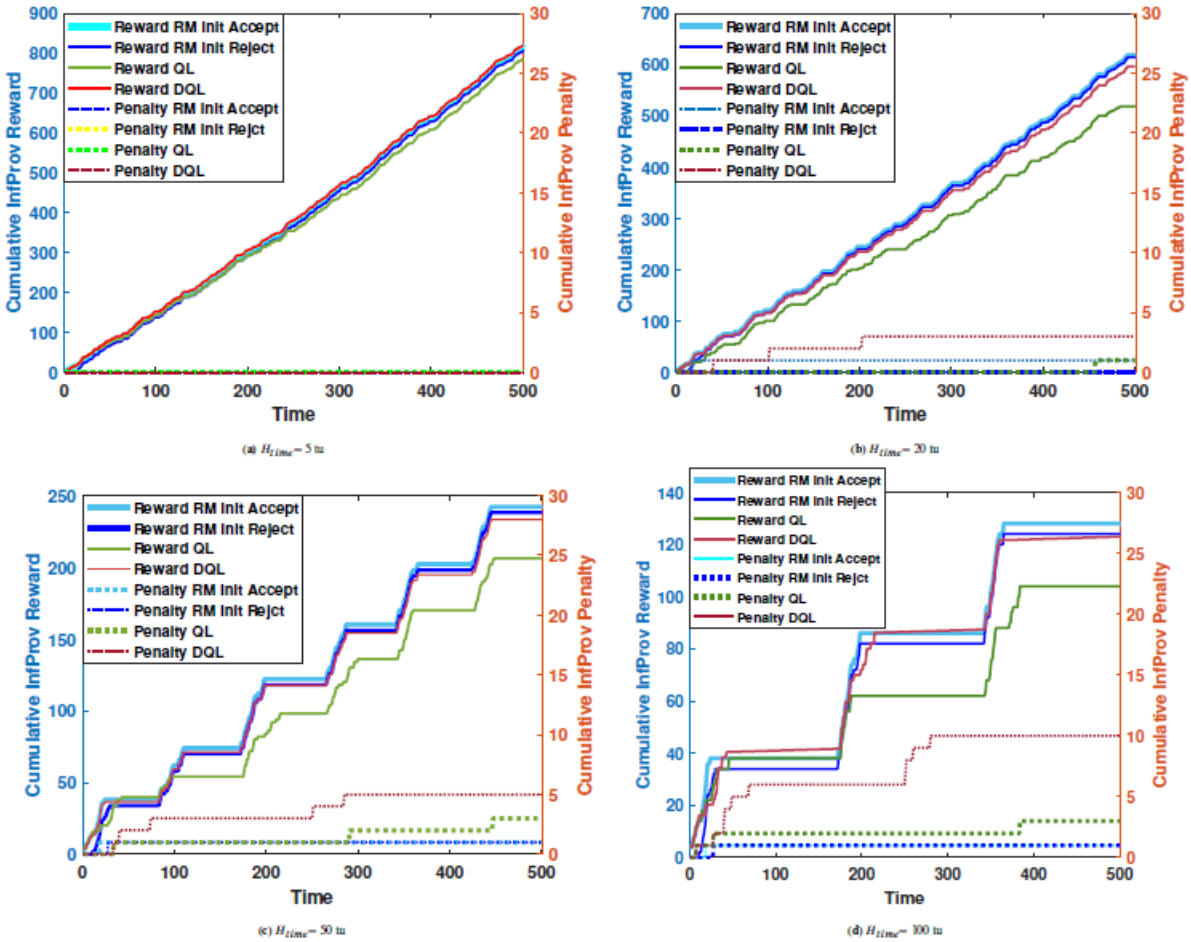


Figure 11 InfProv reward and penalty vs. Time for slice arrival request rate=10

Besides, penalties (accepting a slice without having a resource) occur when the InfProv resources are saturated, and the SAC keeps accepting arrival slices. We remark that penalties are higher when both the arrival rate and the holding time are high, which is evident as the resources are quickly saturated since accepted slices stay longer in the system. Further, we note that most of the algorithms require some time to detect that the resources are saturated and keep accepting requests until the rewardstarts to be negative. Consequently, they change the policy to reject. The time to detect that the resources are saturated is a criterion to understand which algorithm performs well, and hence learned the system's behavior. In this case, we remark that RM obtains the best performances with accept policy, followed by DQL. QL achieves the worst performance. We argue this by the fact that RM, thanks to its regret formula, detects quickly that the reward starts to be negative, and adapts the policy accordingly. Further, DQL with the neuronal network can learn and predict better when to change the policy, compared to QL,where the Q-tables cannot predict when to update.

On the other hand, when the arrival rate and the holding time are low, the probability of having penalties is very small or evennull as there are always available resources to accept new slices (Figure 10).

The results of Figure 10 and Figure 11 also show that in terms of rewards, the QL algorithm is the worst algorithm of all tested algorithms regardless of the arrival rate of each type of slices, by achieving the lowest cumulative reward. This means that it does not learn well when the policy should change from accepting to rejecting, or the contrary. Indeed, QL derived policies that favor rejecting slice requests.

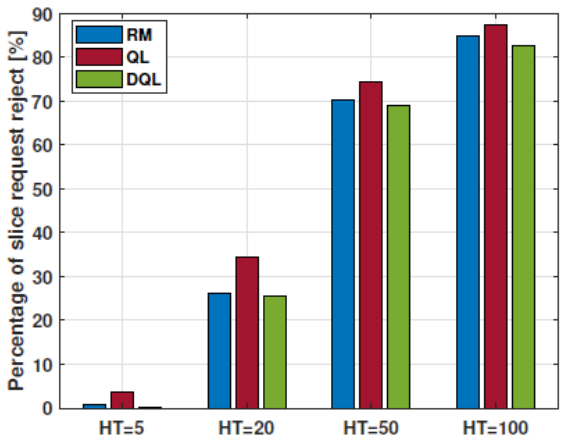


Figure 12 Percentage of slice request reject vs H_{time} for slice arrival request rate=10

Figure 12 presents the percentage of rejected requests according to H_{time} when using the proposed algorithms: RM with accept policy, QL and DQL, and request arrivals rate= 10 corresponding to a frequent slice request arrival. We notice that increasing H_{time} leads to an increase in the percentage of rejection for all the algorithms. This is obvious as high values of H_{time} mean thatadmitted slices will stay longer in the network, and hence low resources are available to accept new slices (i.e., increase the reject rate). Moreover, we noticed that QL rejects more requests than RM and DQL for all the H_{time}, which confirms the low cumulated penalty and reward of QL shown in the two precedent figures.

In Figure 13 we present the percentage of accepted slices according to their type and for different H_{time}, when using RM algorithms with accept policy, QL and DQL. Here our objective is to verify whether the proposed algorithms satisfy the slice priority condition, i.e., the ability to give priority to uRLLC slices compared to the other slice types. We can see that all algorithms favor the uRLLC slice over the other slice types. Further, we see that DQL and RM show the highest percentage of accepted uRLLC slices compared to QL. In other words, these results validate our reward function and the fact of using the price to pay as a wayto enforce priority among the three types of slices.

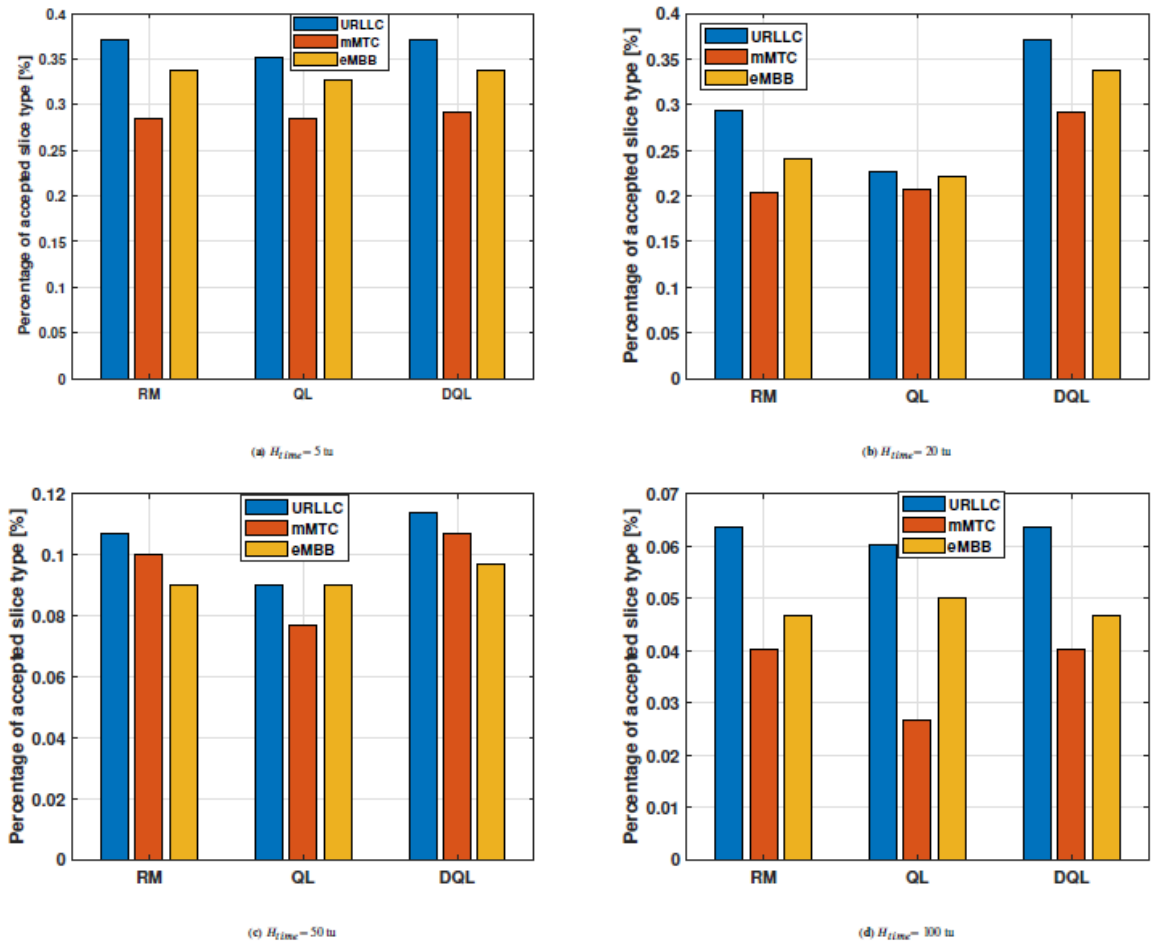


Figure 13 Percentage of accepted slice type for slice arrival request rate=10

One of the biggest challenges when using Reinforcement Learning to solve SAC's problem is whether online or offline learning is better? And what is the time of convergence? So far, we have seen that RM, a fully online algorithm, works well for SAC'sproblem, while DQL and QL need to be trained before being used. Regarding DQL we wanted to understand if an online versioncould make sense to address the SAC problem efficiently. To this aim, we draw in Figure 14 a comparison between the offlineDQL (when we first perform the training phase and then the tests) and the online DQL in terms of average reward. The onlinetraining phase of DQL varies between 0 and 1000 episodes. Each episode represents one training epoch during which the system receives 100 arrival slice requests. The maximum number of episodes depends on the convergence of the online learningmodel. We have increased the number of episodes and calculated the corresponding average reward. We have stopped when thelearning model starts to give a constant reward (between 500 and 1000). Note that the average reward is an average value of thereward obtained for the four considered H_{time}. We clearly observe that the online DQL needs time i.e., more episodes to con-verge (improve the learning) and start achieving the same performance as offline DQL (around 400 episodes). This means thatduring this

period, i.e., before converging, the DQL performances are awful and can seriously affect the business of InfProv. All the results confirm that one of SAC's best policy is to accept whenever the resources are available to maximize the InfProv profile. In this context, RM with accept policy achieves the best performances by reducing the penalty and increasing the reward. DQL and QL could be a good candidate, but there is a need to well tune the learning steps in order to anticipate when the policyhas to change. Indeed, RM uses a simple and efficient formula to understand the need to change policy, while DQL and QL needto learn this. However, in a more complex system, where a high number of actions are available, things may change as RM canhardly, by using a simple formula, capture the behavior of the system. In contrast, DQL can be a powerful solution. But, in the case of a SAC with only two available actions, RM with the accept policy is the best alternative for InfProv.

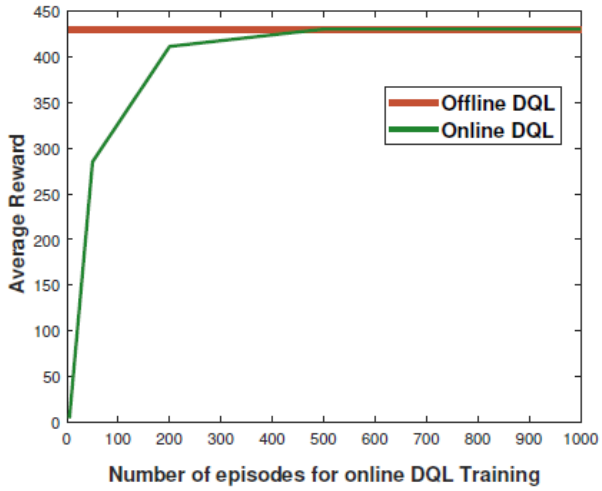


Figure 14 Offline and Online DQL Average Reward of Htime= (5, 20, 50, 100) for slice arrival request rate= 10

5.1.2 MODEL-BASED ADMISSION FOR THE CLOUD DOMAIN

A lot of state-of-the-art research has employed traffic forecasting to drive admission control policies for network slices and for user service requests. However, the prediction values used are usually expected to yield overbooked resources values in an undesirable degree [28], which incurs in costs either for the Infrastructure Provider (InP), due to reduced resource efficiency, or to the tenant (since its paying for resources it does need).

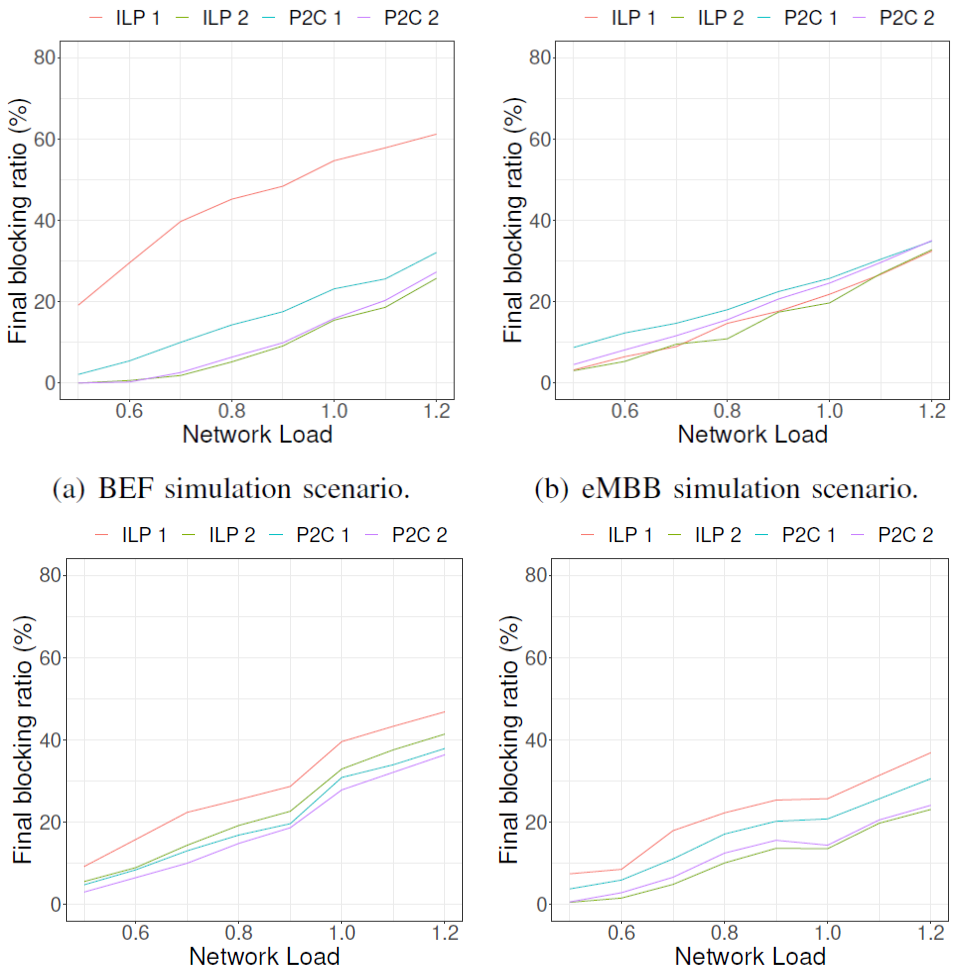
Using Context Aware Traffic Predictor (CATP), a traffic forecasting technique developed within MonB5G, as the traffic prediction mechanism increases the quality of the prediction values. Then, driving an admission control using CATP as the traffic predictor we believe it will yield important improvements over state-of-the-art research for admission control. Coupled with the utilization of CATP, it will be possible to differentiate between reservation of resources by the slices and the real resource utilization, therefore giving a more precise vision to the InP about resource utilization, that can allow it to overbook resources based on the CATP forecast, potentially allowing an increase in the acceptance ratio of the admitted slices and/or the user service requests. This also increases even more the resource efficiency utilization, increasing revenue.

At this point, the development of the slice admission control with CATP forecasting is on the way. The main consideration is to model the Admission Control problem as a type of Time-Window based Bin-Packing Problem with CATP forecast. In short, we would consider a set of network slices over a physical infrastructure, and model the traffic load at specific time windows with variable duration as user service requests (USRs) during that time window. As a first step, the admission control will be static, only allowing USRs for which there is no conflict during the time windows, therefore *avoiding congestion and SLA faults*. The evaluation of this approach it is not in a definite stage, and some additional considerations will be defined as it undergoes progress.

This approach is currently being developed for driving the admission control at the BS station level (RAN domain). It is important to note, that at this stage, we are not considering the problem of VNF-FG placement or network embedding. At this stage, we are focusing on Admission Control based on resource forecast and utilization.

We propose efficient online heuristic algorithm to solve network slice admission optimization problem. We rely on an approach called the "Power of Two Choices" to build the heuristic algorithm. The modeling proposed considers the physical network nodes resources (CPU, RAM) capacities, link resource capacities (bandwidth) and also Edge-specific and URLLC constraints (user location, E2C latency) to calculate the best location for placing the VNFs of the network slices and which network paths to use to chain them.

The algorithm optimization targets are resource utilization minimization and network slice acceptance ratio maximization. To calculate an accurate network slice admission decision is not trivial. The physical substrate network is heterogeneous that is contains different types of DCs with different capacities: Edge Data Centers (EDCs) as local DCs with small resources capacities, Core Data Centers (CDCs) as regional DCs with medium resource capacities, and Central Cloud Platforms (CCPs) as national DCs with big resource capacities. Also, different classes of network slices with different requirements in terms of resources, QoS and coverage area are requested. In this context we propose an intelligent network slice admission policy to offload edge data centers since these are critical resources.



(a) BEF simulation scenario.

(b) eMBB simulation scenario.

(c) URLLC simulation scenario.

(d) MIX simulation scenario.

Figure 15 Comparison of blocking ratios vs network load for two ILP algorithms and two versions of the proposed heuristic: the one adopting random server selection (P2C1), the one adopting the intelligent server selection policy (P2C2)

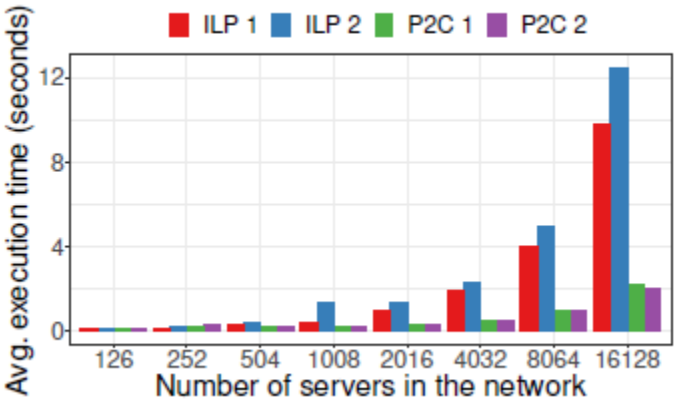


Figure 16 Average execution time evaluation

Figure 15 and Figure 16 present evaluation results recently published in the conference paper [67] which shows the good performance of the heuristic that solves the problem in few seconds under a large-scale network scenario and considering multiple network slice request types: Best-Effort (BEF), Enhanced Mobile Broadband (EMBB) and Ultra-Reliable and Low Latency Communications (URLLC). The heuristic also improves the acceptance ratio of network slices when compared against a deterministic online Integer Linear Programming (ILP) solution.

5.1.3 REINFORCEMENT LEARNING ADMISSION FOR THE CLOUD DOMAIN

As explained in Section 3.1, the admission control problem is closely related with a form of resource management. As the number of slices and/or USRs get admitted into the network, the load on the infrastructure increases as well, and resources are allocated to support the load. However, as the load varies over time it will be necessary to re-allocate the initial resource assignment, and to implement dynamic mechanisms to re-shuffle resources in an efficient way.

In order to provide a solution for this, our CATP-based Admission Control mechanism together with an RL-based mechanism can be deployed in the following ways:

1. Use it as the main solution for admission control. Admission control can be modelled as a combinatorial optimization problem, usually, as a variation of knapsack problems (but other modelling options are viable). Since RL algorithms have proven to be the effective in solving these types of problems, then providing a solution for admission control using RL is an attractive solution
2. Use RL for fine-grain optimization for resource reallocation among the admitted network slices exploiting forecast information with CATP.

When using RL for fine-grain optimization upon admitted slices, the issue will be to decide from which slices/USRs to reclaim resources from and to which slices/USRs re-allocate them to. This RL algorithm will reclaim and reallocate resources at a smaller magnitude when compared to the resource allocation for initial

deployment of slices and USR acceptance. The optimization range of the RL algorithm should be in some allocation region around the initial allocation.

The motivations behind using RL for fine-grain optimization, beyond the main admission control mechanism, can be summarized in the following:

1. The convergence time of RL mechanisms make them inviable for real system deployment due to the huge operational costs they can generate when determining their action policies. Implementing them for fine-grained optimization prevents from incurring those costs of initial deployment and learning.
2. Prevents the system from falling into unstable states as the RL mechanism develops its policy.
3. Action exploration remains an important part of RL mechanism, even after convergence. Even though explorative actions generate benefits to improve and optimize the policy even further after convergence, this explorative nature can translate into undesirable costs for a communication infrastructure.

In order to tackle these issues, the possibility of allowing some form of feedback between the RL fine-grain optimization and the forecasting-based admission control mechanism will need to be provided, similar to [28]. However, the development of these approaches for MonB5G are still under evaluation.

In Section 7.1.1, we have developed an RL approach which learns how to online solve a Binpacking problem. For the admission control that is especially interesting, as our inter-slice orchestrator takes into account the cost of having open server nodes. Essentially, the algorithm tries to place all the VNFs to as least server nodes as possible. There is a two-fold advantage in that: a) having less servers on saves energy, b) fitting VNFs to the least amount of server nodes, allows for potentially very large slices that will request to be placed in the network, to find available resources; these resources would otherwise have been rejected service by the operator.

We propose a Deep Reinforcement Learning (DRL) approach based on the Asynchronous Advantage Actor Critic Algorithm (A3C) introduced by [68] to optimize network slice admission. The approach leverage on the results of [69] that has shown that A3C yields good results in terms of acceptance ratio in the domain of to the Virtual Network Embedding (VNE) problem.

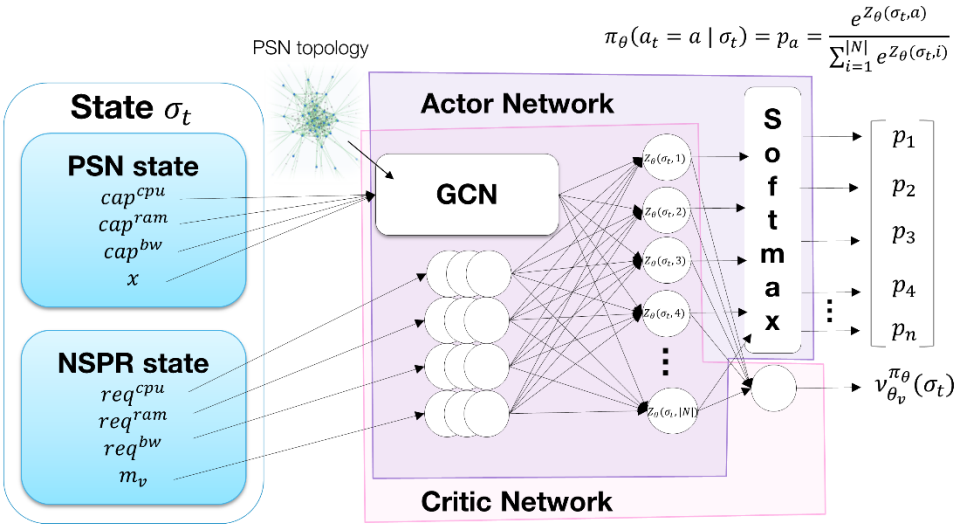


Figure 17 Architecture of the proposed DRL agent

Figure 17 presents the architecture of the proposed DRL agent. The agent maintains two neural networks that are trained at the same time, one that generates the action selection policy called Actor Network and one that will calculate the state-value function estimation at each time step *t*. To reduce the action space, we divide the calculation of the admission decision in several steps. Each step corresponds to the admission of one specific VNF *v* of the Network Slice Placement Request (NSPR) and mapping of the virtual links associated with VNF *v*. A training episode will therefore correspond to the admission of the NSPR into the Physical Substrate Network (PSN) or its complete rejection.

At each training step the Graph Convolutional Network (GCN) layer is fed with the features of the PSN nodes: the avaliable CPU capacity (*cap_cpu*), the avaliable RAM capacity (*cap_ram*), the maximum outgoing bandwith (cap_bw) and a placement mask (*x)* indicating the number of VNFs of the current NSPR already placed in each node. The features representing the resource requirements of the VNF *v* to be placed are separately transmitted to a fully connected layer. We consolidate the outputs of its two layers into a second fully connected layer which has a number units equal to the number of nodes of the PSN. This layer is connected to a last neuron which is used to calculate the state value and to a softmax layer which calculates a probability *Pa* of placement in each node *a* of the PSN, which corresponds to the placement policy π.

At the end of a training step there is an attempt to place the current VNF *v* on the server with the highest probability of placement value. The reward obtained and saved and there is a PSN updated. At the end of an episode, a loss function based on all the rewards and state values obtained is calculated and an update of the weights of the neural networks is performed using the gradients of this function.

5.2 Multi-domain slice admission control algorithms

5.2.1 MULTI-DOMAIN CENTRALIZED ADMISSION CONTROL USING REINFORCEMENT LEARNING

The forecasting-based Admission Control mechanism together with the RL fine-grained optimization are in principle technological-domain agnostic. But for an end-to-end implementation and evaluation, it is necessary to get load traces at different technological domains, and this data has not been made available. In order to develop this part, we would rely on the testbed and getting the necessary data from it. However, this is still heavily under evaluation, and developing a clear mechanism for this is not possible at this point.

So far, we tackle the problem of admission control in multiple domains by applying the DRL agent introduced in section 5.1.2 in a multi-domain network context. The multi-domain physical substrate network (PSN) is represented in Figure 18. The PSN is divided in three parts: the Virtualized Infrastructure (VI), the Access Network (AN) and the Transport Network (TN). The virtualized infrastructure represents the set of data centers (DCs) interconnected by network elements (switches and routers) and located at Point of Presence (PoP) or centralized. They offer IT resources to run VNFs.

We define three types of DCs with different capacities: Edge Data Centers (EDCs) as local DCs with small resources capacities, Core Data Centers (CDCs) as regional DCs with medium resource capacities, and Central Cloud Platforms (CCPs) as national DCs with big resource capacities. The Access Network (AN) represents User Access Points (UAPs) (Wi-Fi APs, cellular, etc.) and Access Links. Users access the slices via one UAP, which may change during the life time of a communication by a user. The Transport Network (TN): represents the set of routers and transmission links needed to interconnect the different DCs and the UAPs.

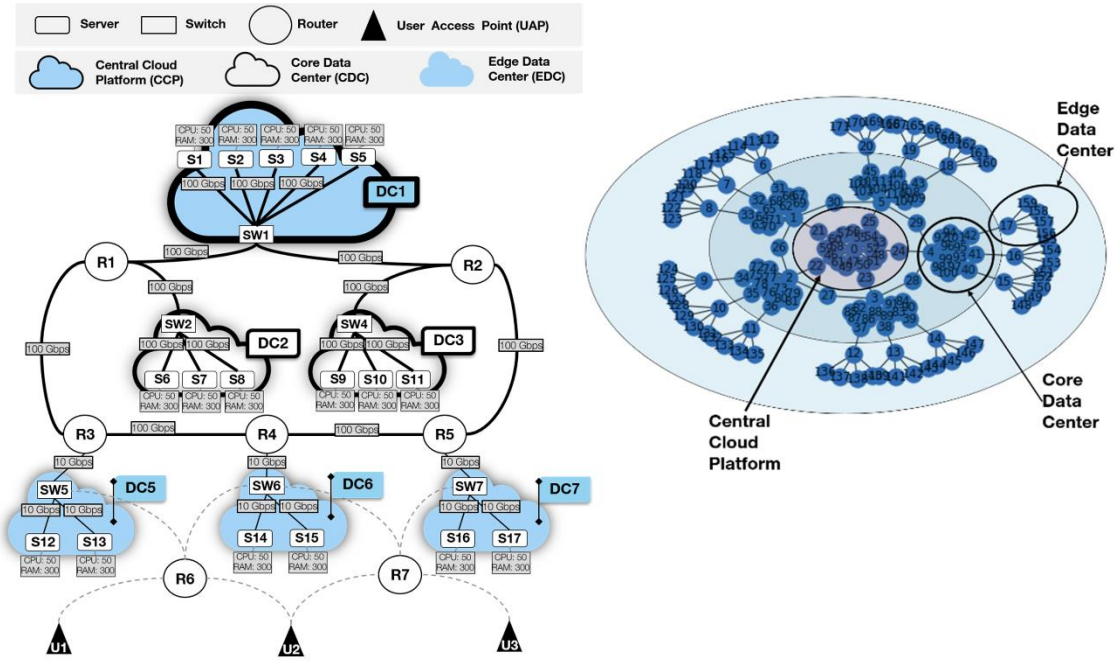


Figure 18 End to end multi-domain physical substrate network modeling

6 Intra-slice Orchestration

To focus on slice orchestration, this Section gathers the challenges and proposed methods for AI-based intra-slice orchestration relying on the architecture, principles and methods introduced in Sections 4 and 5 above.

We highlight: i) domain-specific mechanisms for intra-slice orchestration as per the specificities of the architecture domains and the use of AI methods for configuration, migration, and scaling applied to VNFs as main components of a network slice; and ii) cross-domain intra-slice orchestration with an E2E approach including the use of AI methods for VNF chains placement and scaling with regards to the application of distributed algorithms and multi-agents methods for multi-domain network slicing.

It is important to mention that when we consider single-domain algorithms for intra-slice orchestration, these domains and associated contributions are not standalone contributions but are to be considered as building blocks that will be integrated as components of the distributed architecture discussed in Section 4 above.

6.1 Domain-specific intra-slice orchestration

In this sub-section, we first present the set of AI methods we are considering for VNF configuration, reconfiguration and migration when it comes to domain specific intra-slice orchestration. Then, we go through the AI methods used for intra-slice orchestration.

6.1.1 AI METHODS FOR VNF (RE-)CONFIGURATION AND MIGRATION

Modern 5G networks consist of a huge amount of distributed Virtual Network Functions (VNFs) between multiple geographical locations. Manually managing these VNFs and their data-flows to provide the maximum performance at minimum cost is impossible. Since the world is already moving towards data-driven automation, Augmented Intelligence (AI) algorithms can be employed to tackle the issue of zero-touch VNF placement and migration.

There are several AI methods that have shown promising results for VNF placement and migration. Past projects mostly focused on traditional optimization algorithms, like Integer Linear Programming (ILP), to identify the placement that maximizes - or minimizes - a specific metric. However, the optimization algorithms' complexity and calculation time increases exponentially, posing problems with scaling in larger networks. Newer works focus on exploring the abundant resources of network data available to train Deep Neural Network (DNN) based algorithms to forecast the best VNF placement -or reconfiguration- given the current or historical data. An important breakthrough was the use of time-series forecasting Long Short-Term Memory (LSTM) and spatial data identifying Convolutional Neural Networks (CNN) algorithms to identify trends and patterns that trigger a specific VNF reconfiguration that is predicted to perform the best given a metric. Compared to ILP, these algorithms are able to find the best configuration, with the least computational cost which is important for real-time, dynamic systems.

Since networks can be modelled as Markov Decision Process (MDP) environments, Reinforcement Learning (RL) algorithms demonstrated their ability to dynamically tackle the issue of VNF reconfiguration and migration. This field is inspired by the recent breakthroughs in self-driving cars, robotics, and the revolution that OpenAI Gym environments sparked. Recent projects are focused on dynamically solving the VNF migration problem with cutting edge algorithms, currently exploited by other industries also, such as Deep

Deterministic Policy Gradients (DDPG), various Actor-Critic Methods (A2C, A3C), and Proximal Policy Optimization (PPO).

Recently researchers started to exploit the repetition in networks by employing Multi-Agent RL (MARL) algorithms to achieve better learning in parallel with multiple algorithms that share the same problem and cooperate to solve this issue. This solution is faster, more efficient, and with less cost.

New scalable cooperative MARL algorithms with multiple goals or objectives and complex shared rewards emerged to tackle the dynamic VNF reconfigurations and migration issue, but this time also in great scales.

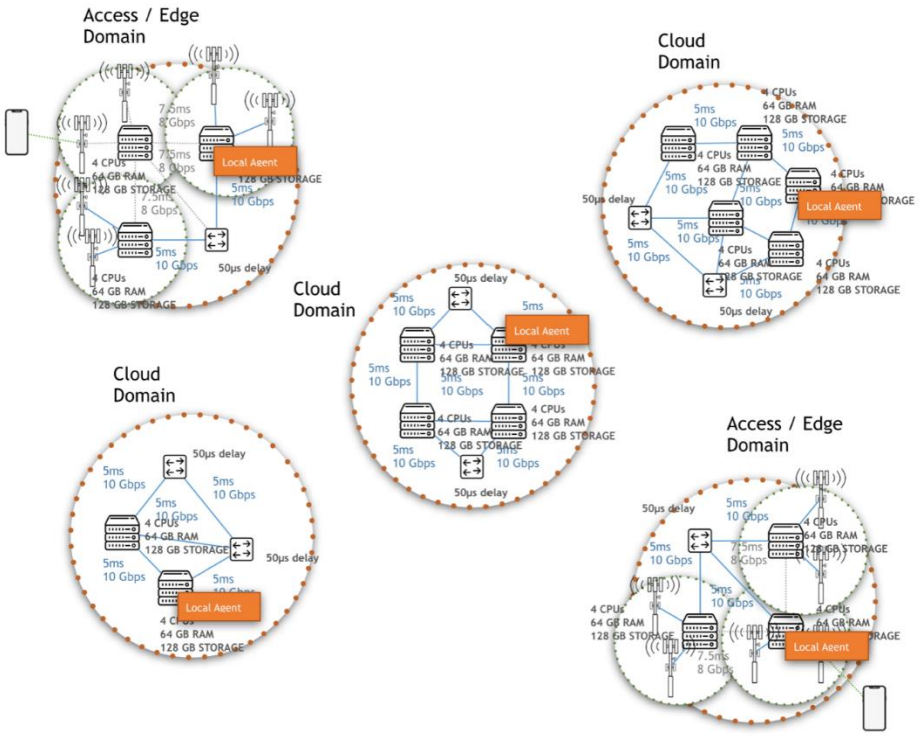


Figure 19 Multiple geographical domains with local agents instantiated

We recommend the use of discrete MARL algorithms, such as Multi-Agent Deep Q Network (MADQN), to efficiently solve the problem of dynamic VNF reconfiguration and migration in a scalable way. The large placement state space can be shared between multiple agents that perform the VNF placement in parallel.

The entire network can be split into multiple logical or geographical domains. VNF placement agents can be instantiated locally in each domain. The placement agents can communicate with a global agent that is responsible to distribute the VNFs-to-be-placed amongst them. A shared reward function can enable cooperation for the optimization of a specific metric, such as latency, throughput, or a complex metric.

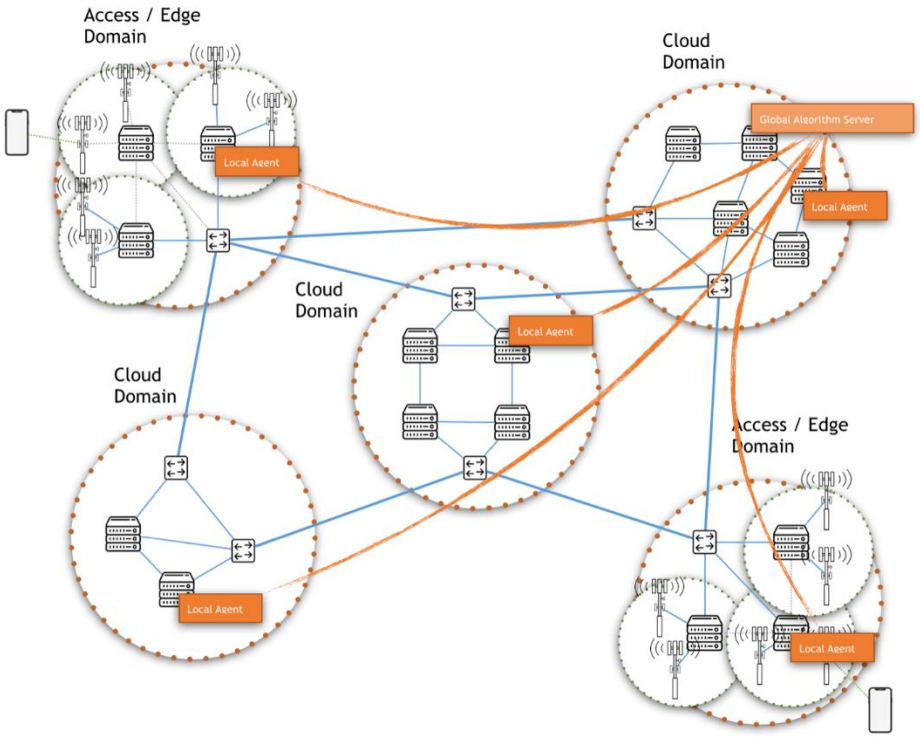


Figure 20 Interconnected domains sharing a common reward to enable cooperation between the agents

The VNF placement is performed based on the capacity and load of the connected links and servers, the congestion level of local and alternative computing resources, and KPI predictors. Local placement agents attempt to identify feasible local reconfigurations, affecting only a small, local part of the network. These reconfigurations could apply a specific policy, without the need for global network VNF reconfiguration. One multi-agent RL-based solution [70] that we are considering in the context of intra-slice orchestration is SafeRL (see Figure 21 below).

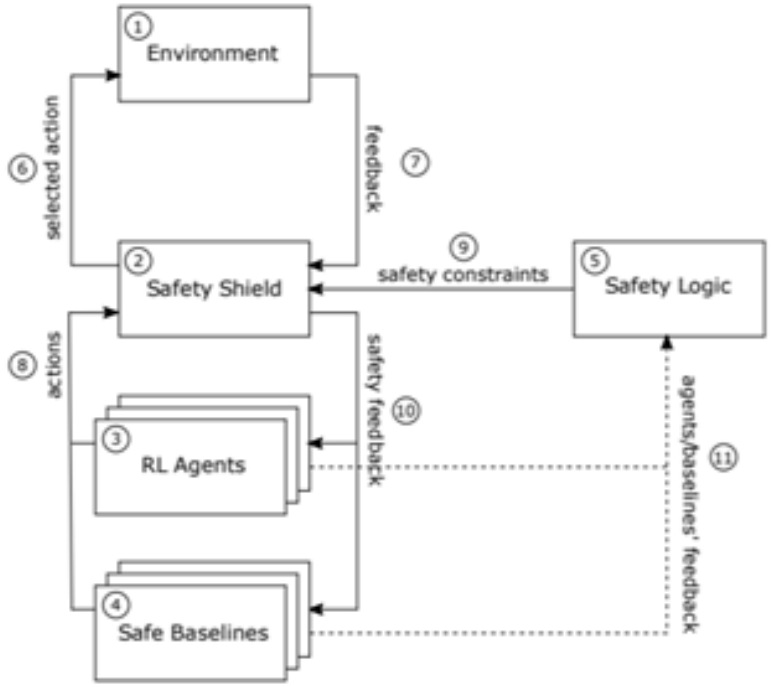


Figure 21 SafeRL with safety shield, logic and multiple safe baselines

This solution combines the RL-based approach with a safety shield, to only allow the RL agents to find new reconfigurations within certain constraints. In the traditional RL model, the agents directly interact with the environment and receive feedback.

In Safe RL, the safety shield acts as a proxy between the agents and the environment to protect it from possibly unsafe actions proposed by the agents. The safety logic collects proposed actions by agents and baselines based on the current state of the environment and chooses a final safe action to be performed on the real environment. The RL agents continuously train based on the feedback of the action that was chosen by the safety logic and executed by the safety shield.

Using different learning methods enables parallel experimentation with different techniques and parameters, to determine the most suitable safe action among them. The safe baselines can be defined as a set of slice configurations that are known to satisfy the safety requirements with acceptable performance (e.g., it may not be a configuration that leads to optimal usage of resources, but it allows for the slice SLAs to be met). The safety logic for the slice reconfiguration case can include slice SLAs and prediction models to estimate how well the proposed action will do with respect to certain performance indicators. The local AE may also be asked for predictions on slice KPIs.

6.1.2 *AI METHODS FOR INTRA-SLICE SCALING*

6.1.2.1 DEEP NEURAL NETWORKS (DNN) FOR INTRA-SLICE RESOURCE SCALING

In what follows, we will report our progress on the problem of intra-slice resource scaling in the MEC domain of the network. Before delving into the details and to assist the reader, in the Figure 22 below, we show the setup of our problem.

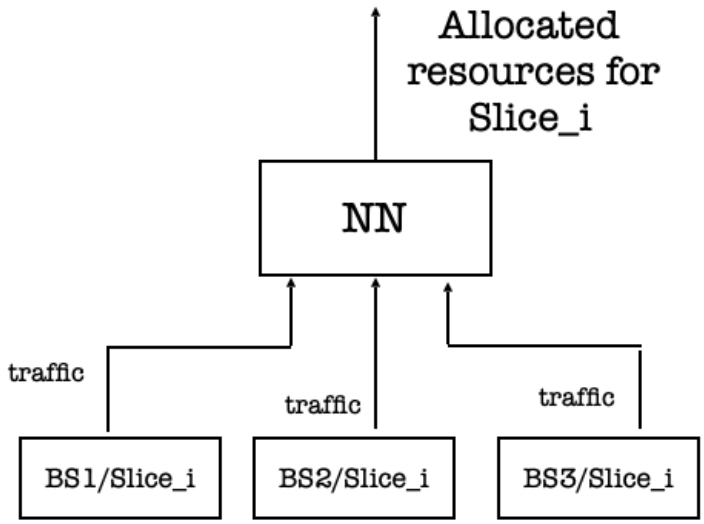


Figure 22 Slice i receives traffic from 3 BSs, and the traffic values are gathered in a Neural Network which requests resources for the next time steps

In the RAN side, we condition our viewpoint on the *i*-th slice (e.g., Instagram) only. The goal is to design an NN that gathers these measurements and allocates resources; this is placed in the MEC, which has higher computational capacity. The target is summarized as follows:

- **Service/Deployment.** At time *t*, receive a number of signals *M* (as many as the BSs), from a time window W (some integer number), and the DNN will return a single scalar value *R* (stands for resource), which is the decision for the resource allocation for the whole slice at time *t+1*, across the multiple BSs that this traffic receives traffic from.
- **Problem Setup.** We approach this problem from a data-driven viewpoint. To do so, we train a long-short-term memory (LSTM) network, which is known to be very effective in applications that include time-series data. We take a step further and instead of simply predicting values with the LSTM, we ask from it to essentially solve an optimization problem, and demand from it to balance three different competing goals:
  - Overprovisioning: SLA violations and under-provisioning may be of critical cost, but also "wasting" resources in order to be safe, is highly problematic and has to be avoided.
  - Under-provisioning: We definitely need the allocated value to be on the "safe side", i.e, reserve slightly more resources than the slice needs, because failing to do so, will be of severe cost to the infrastructure operator; this situation directly implies SLA violations on the contract between the tenant and the operator.

- Reconfigurations: One really important source of costs is the constant and unreasonable change of resources from one timestep to the next, that the $|R(t) - R(t-1)|$. We penalize the DNN and do not allow it to scale up and down at will and assign to the reconfigurations some amount of money, exactly as we do with the rest of the costs.

Notably, this simple NN architecture tackles the same problem as the AZTEC [71], but does so in a much simpler way. We train the network using as batches of inputs (X) the values of the BSs over a period/window W, and the corresponding output is a single scalar (y). Importantly, as the loss function consists of three entities, we need to carefully select the values *w1, w2, w3* that show the emphasis on each of the individual components above.

Among the obtained results, we show here the results for two cases:

a. one where the reconfigurations have the highest penalty rate (i.e., weight), and
b. and second one where under/overprovision rates are relatively much higher than the reconfiguration, with higher penalty to the under-provisioning.

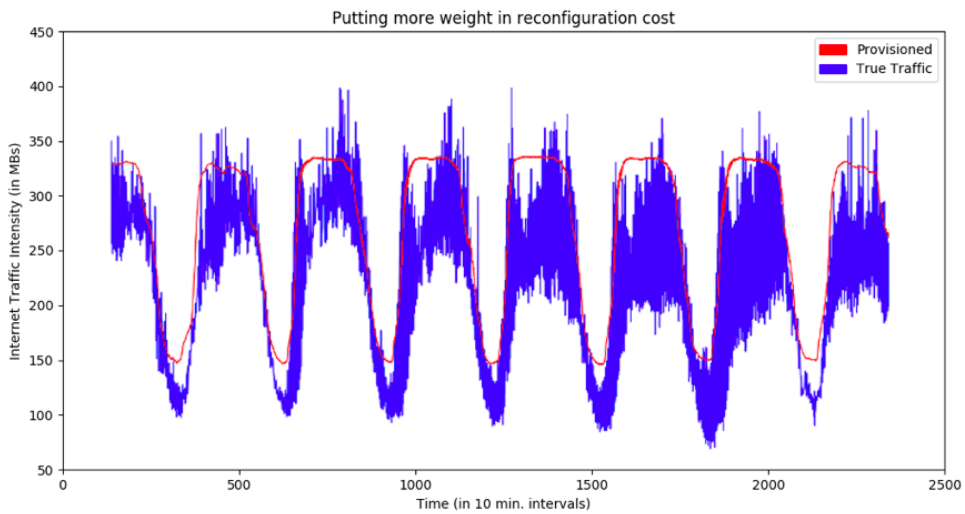


Figure 23 True and DE provisioned traffic intensity vs time for DE with more weight on reconfiguration costs

In the plot of Figure 23 above, we see that for case a) (in blue) the sum of traffic values that data center *j* in the MEC "sees", and with red, we see the value of the allocator.

As expected, since the penalty on the reconfiguration is very high, the allocated resources waveform is quite smooth and does not "fall into the trap" of constantly changing its *R*.

Furthermore, note that since the under-provisioning penalty is higher than the overprovisioning, the allocator simply "learns" the long-term trend, in order to avoid rescheduling resources, and tries to stay higher than the demanded traffic (in blue).

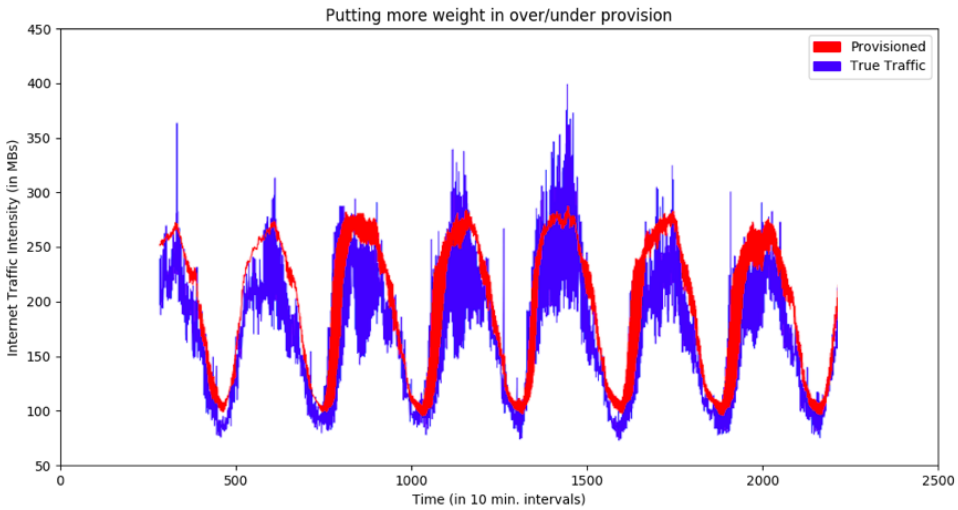


Figure 24 True and DE provisioned traffic intensity vs time for DE with more weight on over/under-provision costs

On the other, right above we see case (the colors represent again, blue: "True" and red: "Allocated") (b): Since the penalty on the reconfiguration is lower, we see a much spikier red/allocated waveform, which comes from the fact that we do not penalize changes too much. Reasonably, the allocated waveform follows much better the true one, and can find a much better trade-off, especially on the overprovisioning side of things. The allocation of plot (b) does not "waste" resources in order not to make reconfigurations, and that is why it achieves a much lower overprovision cost.

6.2 Cross-domain intra-slice orchestration

6.2.1 *E2E VNF AND SLICE PLACEMENT AND SCALING - HIERARCHICAL/CENTRALIZED*

Regarding the E2E VNF and Slice placement, we tackle here the problem of how to admit a maximum number of network slice requests into an E2E physical network infrastructure. This involves solving the E2E network slice placement and resource orchestration optimization problem.

The network slice placement optimization problem contains two main elements: the Physical Substrate Network (PSN) and the Network Slice Placement Request (NSPR). The PSN is represented as a weighted undirected graph Gs = (N, L), where N is the set of physical nodes in the PSN, and L = (a, b) in NxN refers to a set of substrate links. The available CPU and RAM capacities on physical substrate nodes are defined respectively as CAPCPUn, CAPRAMn, n in N. The available bandwidth capacities on the PSN links are defined as CAPBW(a,b), (a,b) in L.

We consider that each network slice is a finite number of VNFs that is placed and chained on the PSN. VNFs are batched and introduced in the network as NSPRs. The NSPRs are similarly represented as a weighted undirected graph Gv = (V, E) where V donates the set of VNFs in the NSPR, and E = (a,b) in VxV is a set of Virtual Links (VLs). The CPU and RAM requirements of each VNF of a NSPR are defined respectively as

REQCPUv and REQRAMv, v in V. The bandwidth required by each virtual link in a NSPR is given REQBW(a,b), (a,b) in E.

We formulate the network slice deployment/placement optimization as follows:

- *Given:* a NSPR graph Gv = (V, E) and a PSN graph Gs = (N, L),
- *Find:* a mapping Gv to Gs =(N1,L1), N1 is a subset of N, L1 is a subset of L,
- *Subject to:* the VNF CPU requirements REQCPUv, v in V, the VNF RAM requirements REQRAMv, v in V, the Virtual Links bandwidth requirement REQBW(a,b), (a,b) in E, the nodes CPU available capacity CAPCPUn, n in N, the servers RAM available capacity CAPRAMn, n in N, the physical links bandwidth available capacity CAPBW(a,b), (a,b) in L
- *Objective:* maximize the network slice placement requests acceptance ratio, minimize the total resource consumption and maximize load balancing.

To solve the proposed network slice placement/deployment problem we introduce a novel algorithm called Heuristically Accelerated Advantage Actor Critic. This algorithm is designed as an extension of the Advantage Actor Critic algorithm introduced by [68] and successfully applied by [55] in the domain of the Virtual Network Embedding (VNE), which can be considered as a variant of the network slice placement/deployment problem.

As explained in [55], the Advantage Actor Critic algorithm when applied to the VNE problem yields good results in terms of acceptance ratio of network slices but it needs a considerable amount of training episodes to achieve that. While the training is not done, the agent keeps taking bad decisions, what leads to rejection of many NSPRS. This fact reduces the safety of using this kind of approach in a practical environment.

To deal with this problem we make use an efficient heuristic based on the Power of Two Choices (P2C) developed in a previous work [66] to accelerate the learning process. In the proposed approach, this heuristic is used as a support to the DRL agent to accelerate its convergence as shown in the Figure 25.

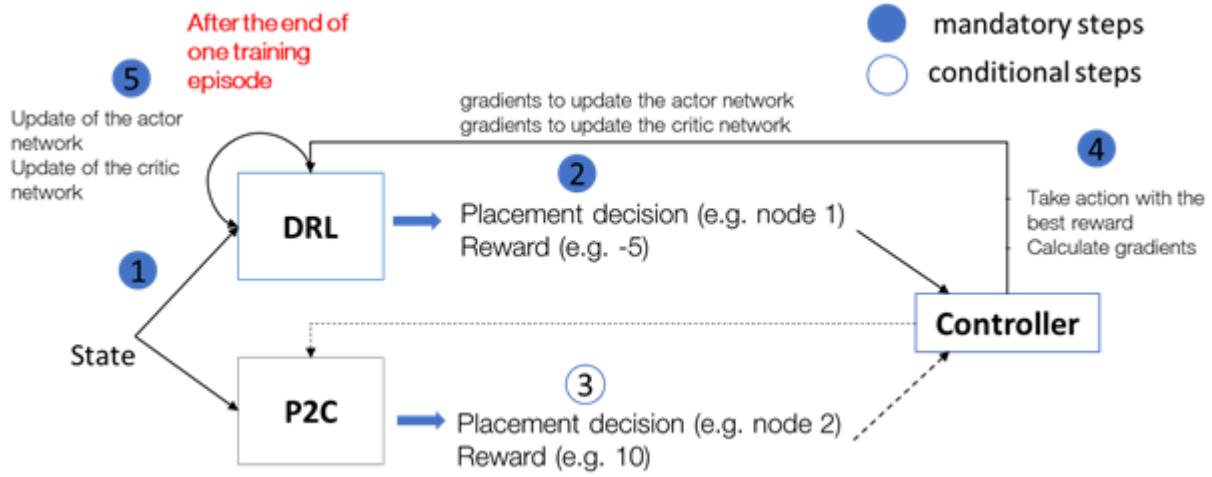


Figure 25 Learning control using previous contribution: heuristic based on the "Power of Two Choices" (P2C) principle

We implemented our approach using PyTorch and ran some initial simulations to evaluate the convergence. We considered the EMBB simulation scenario as described in [66] with a network load of 90%. We show in the Figure 6.8 the Acceptance Ratio of the Advantage Actor Critic (DRL) and the Heuristically Accelerated Advantage Actor Critic (DRL+P2C).

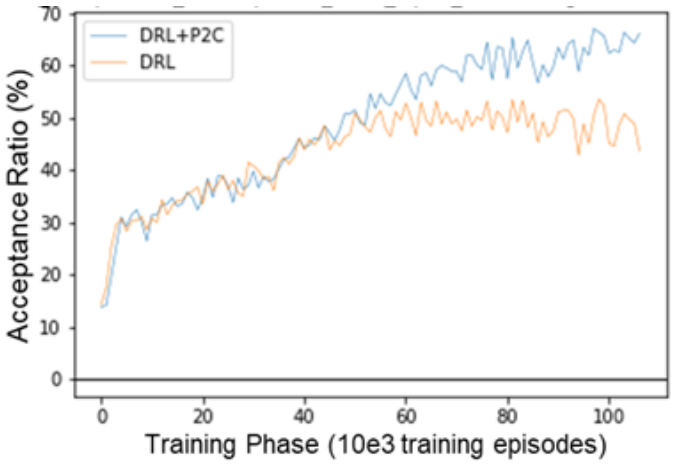


Figure 26 Network Slice Acceptance Ratio vs Training Phase

The preliminary results in Figure 26 show that the use of the P2C helps to improve the DRL performance. As next steps, we plan to extend this approach to also handle QoS constraints like latency requirements.

6.2.2 *DISTRIBUTED ALGORITHMS AND USE OF MULTI AGENTS REINFORCEMENT LEARNING*

MARL algorithms and specifically Multi-agent Deep RL, had an outstanding evolution and gained more traction in more industries recently. Learning in MDRL algorithms is fundamentally more difficult than single-agent RL due to the issues regarding the collaboration between the agents.

MARL algorithms can be divided in multiple categories and each implementation can be mixed with different techniques, from single-agent RL or other AI implementations. MARL algorithms can be either cooperative, competitive or operate in mixed scenarios. The agents can solve goals, maximize or minimize rewards or have inputs that are shared or independent amongst the agents. To enable VNF placement between multiple geographical domains, the most important case is cooperative learning and is usually achieved through a common complex weighted reward.

We plan to extend the approach introduced in section 6.2.1 to use it in a multi-agent architecture as shown in Figure 27. We take as basis the multi-domain non-cooperative VNF-FG embedding architecture introduced by [38] and apply it to our case, using the DRL agent based on HA-A2C algorithm. This will allow our approach to scale up even further as we consider in this multi-agent scenario that we would have one DRL agent per network domain. Each DRL agent will learn on how to make network slice placement decisions in his

respective domain. A higher-level decision engine is responsible for evaluating, when needed the decisions taken by each local DRL agents and to choose the most advantageous.

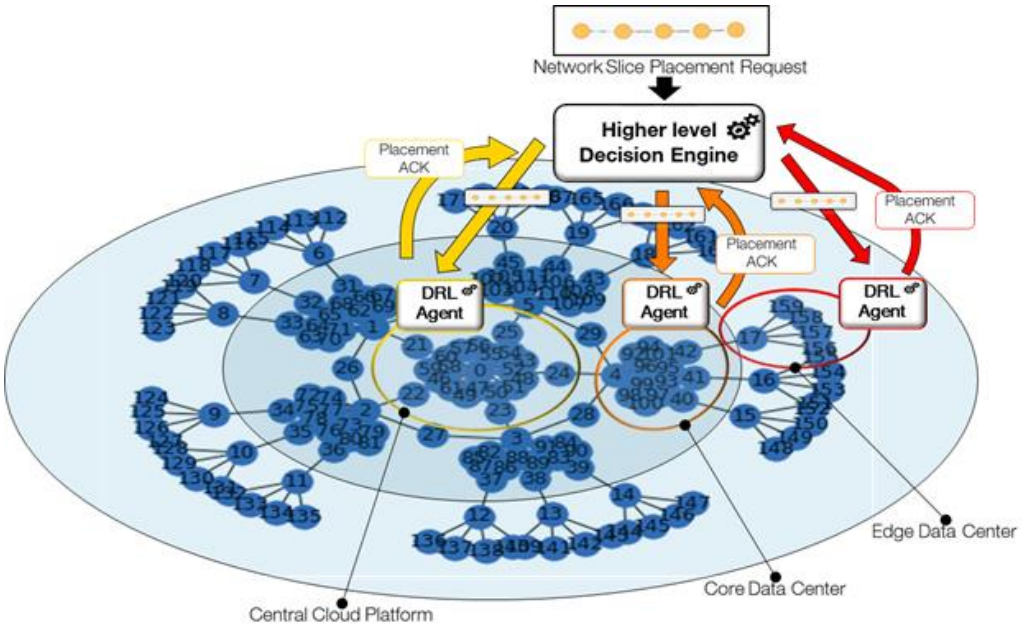


Figure 27 Multi-agent DRL approach for cross-domain slice orchestration

7 Inter-slice Orchestration

After having discussed the problems of slice admission and intra-slice control/orchestration, this final technical section focuses on the problem of inter-slice orchestration. The said problem lies a level above the intra-slice one. As an example, when a specific slice demands a huge increase in resources, as the inter-slice DE has full view of the infrastructure at its disposal, it can move the aforementioned slice to a fresh dedicated server node, so that the slice KPIs remain guaranteed. Essentially, the inter-slice DE is the entity having a wider picture (supervision) of the network operations and its performance.

At large, its role is to coordinate the VNF processes across all technological domains inside the physical network of the operator. The said coordination has to result in a *robust operation* of all slices (i.e., respecting the SLA agreements and guaranteeing promised KPIs) and this has to happen in the *most cost-efficient* way possible for the operator. The problem is highly complex, because if the operator (inter-slice DE) starts moving VNFs around at will (even if the taken decisions seem correct in the short run), this may cause a huge damage to the KPIs of the slice tenants in the long-run. These reconfigurations will be followed by increased end to end delays (or even severe congestions), as packets may start gathering while the inter-slice DE moves the VNFs.

Thus, another key objective for the DE at this level is to pay attention to reconfiguration costs. Importantly, the DE at this level implicitly has control over the admission as well. One of the main objectives of MonB5G is to be able to host a large number of slices, and to this end, the inter-slice orchestrator has to take into account this dimension as well. Therefore, although we need to respect SLA agreements, another key goal for us is to make sure we *do not* waste resources by operating unnecessarily too many server nodes open. Having the minimum amount of physical infrastructure nodes OFF (a) is environmentally friendlier, (b): less costly for the operator and (c): allows to host more slices in the future.

To bind the intra-slice together with the inter-slice DEs, we present here a figure that shows conceptually how these entities coordinate and the distributed nature of the decisions taken.

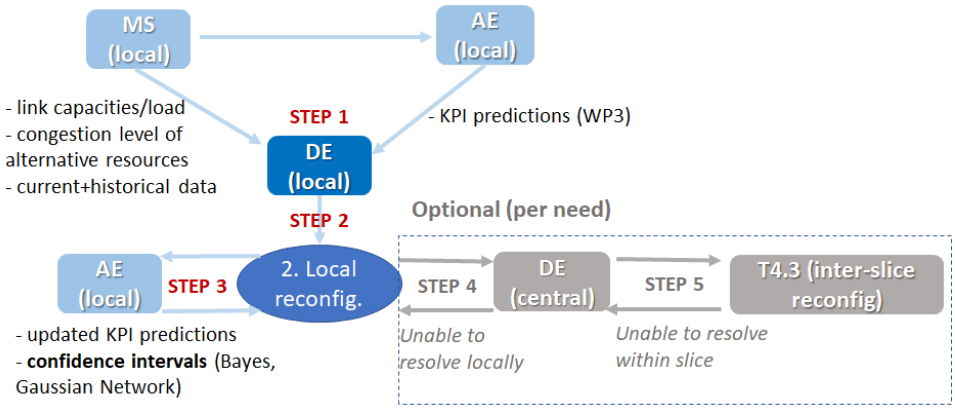


Figure 28 Conceptual interaction of intra and inter slice DEs

Notably, this should not be taken as an alternative DE architecture, but rather as a conceptual scheme of how the algorithms of Section 6 and Section 7 relate to each other.

In what follows, we will see some approaches related to domain-specific and cross-domain orchestrators.

7.1 Domain-specific inter-slice orchestration

At this level, the inter-slice DE is responsible for reconfiguring the slice-server assignment only on one technological domain (e.g., RAN or MEC or transport). It is important to note here, and for the remained of this section, that when we describe some solutions as "domain-specific", for example, an algorithm targeting the RAN technological domain, we assume that this solution is not a standalone algorithm, but will be integrated (in the remainder of the project) into the end-to-end DE architecture described in Section 4, interacting through the defined interfaces with other DEs (and locally residing algorithmic components).

7.1.1 Q-LEARNING WITH MULTIPLE OBJECTIVES

We need to carefully define the goal, i.e., objectives, and the model we assume before continuing to the proposed algorithmic solution. We employ here a simple, yet widely used model for the slicing problem. A pictorial representation of what we assume as a slice and the corresponding network infrastructure is presented below in Figure 29 [13].

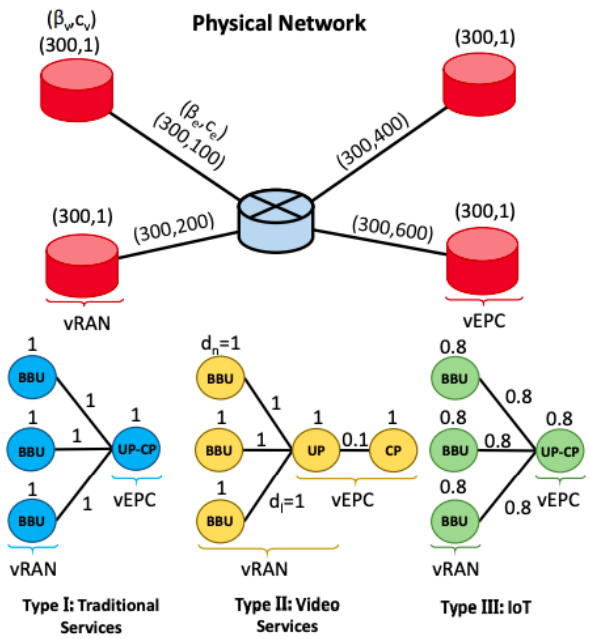


Figure 29 On the top: Network operator infrastructure resources, on the bottom: Different slices as graph embeddings, in all domains

Moreover, we are considering a decoupled version of the network and focus only on one side of it, e.g., the RAN or the MEC or the transport. Thus, our analysis and solution would hold for an interslice orchestrator at any technological domain of the network for the objectives we choose to optimize.

Slice and Infrastructure: Modelling and Problem Setup. According to the Figure above, each slice consists of some virtual network functions (VNFs) or processes, and depending on the type of VNF, these can be placed either in the radio access network (RAN) or the MEC. In our first attempt to solve this problem, we will assume a simplified version of the architecture we see above. We are dealing with a set of slices $L = \{1, \ldots, |L|\}$, where each slice comes with one VNF only. Note that each VNF (and therefore slice) comes with some traffic intensity D_i, for all i set of slices L, which is a scalar real value.

The infrastructure of the operator is assumed to be a set of server nodes $N = \{1, \ldots, |N|\}$ that have some specific capacity C_i, with i in set N, and each can fit inside some VNFs. Importantly, each VNF (and therefore slice) comes with some traffic intensity D_i, which we assume to be random. Finally, must be always placed in one node only. Put differently, it is prohibited to split a process into more than one server nodes.

Time Scale. In this basic setup, the time is slotted and discrete; the system starts from $t = 0, 1, \ldots$ and evolves infinitely. In other words, we face the problem as a non-episodic process. Usual examples of episodic Markov Decision Problems (or Reinforcement Learning (RL)) in the unknown environment case) are the ones that have a fixed (deterministic or random), and finite horizon. We view our problem as an "ever-learning" infinitely long RL episode. At each time, the agent/learner is able to fully observe the state and the rewards of the system, and is also able to take an action.

State Space. A reasonable first thought for a state space is to take all the possible configurations/placements for the slices in the possible nodes/links that are available. The configurations are basically all the possible r_{ij} with i in L (for slices) and j in N (for nodes) where each slice is assigned to only one node. For each slice, we can have N possible placements, therefore we have N x ... x N, L times (i.e., N^L), as many as slices we have. However, if we can control deterministically the evolution of the slices configurations, then there is nothing random in the transitions of the system. The only random entity in this process is the slice traffic demand. In Markov process terms, this could easily be represented as a probability transition matrix with entries d_{ij} with i in D and j in $D = \{0, \ldots, |D|\}$ being the possible values the traffic demand can take. Note that there are L underlying such probability transition matrices, *one for each slice*. This accounts for N^L x D^L states in total.

Action Space. A crucial entity of the MDP/RL framework is the action space of the agent. Here we assume that the agent's control is the configuration of the system, so the action taken by the learner at time step t, ends up being the part of the state that corresponds to the slices configuration at the immediate next time instance (at t+1).

Importantly, we have to specify the KPIs of interest that characterize a high-performance inter-slice orchestrator.

- ***Free up Resources:*** One first problem is the one where the objective is to maximize the free space of the resources so that when a new slice wants to be embedded in the PN, the operator is able to fit it.
- ***SLA Violations:*** Given the current demand and the configuration, new slice demands are revealed at *t+1*, and we pay some penalty that depends on where we placed the VNFs, and on the demands that arrived (which potentially caused capacity violations on the assigned server nodes at *t+1*). These

penalties model some sort of SLA violation, since exceeding the capacity of the server node directly translates to money paid from the infrastructure owner to the slice tenant.

- ***Reconfiguration Costs:*** Every time instance, the controller agent has to decide whether to change the slice/server assignment. The L1 norm $|X_{t+1} - X_t|$ (where X is the slice/server assignment) expresses the number of reconfigurations we had to make from. If we weigh this quantity with some c_{ij} (that indicates the cost the corresponding reconfiguration), we have the total cost of the reconfiguration of two consequent slice/server assignments.

We stress at this point, that the costs we have listed above, for sure implicitly help the slice-tenant perceived performance, however, they are explicitly network-operator-centred. Essentially zero cost at the above metrics (2, 3) implies optimal experience for the slice tenant. However, since we need one global cost, we have to define the cost as a convex combination of (1, 2, 3) and therefore we can never explicitly set to zero any of the above metrics, but rather only minimize a combination of them. Therefore, for now we omit a crucial slice tenant-perceived performance metric, i.e., the delay experienced by the tenant, which we will discuss in detail later and add to this flexible framework.

Since we now have formalized all the necessary entities (state space, action space, rewards description) of the RL framework, we can now formalize our immediate reward/cost as a function of the current state, the next state and the action taken by the agent.

$$
\begin{aligned}
C^a_{ss'} &= C_1(a) + C_2(s,a) + C_3(s') = \\
&\underbrace{\sum_{j=1}^{N} \mathbf{1}_{\{\sum_{i=1}^{L} r_{ij}(t+1)\geq 1\}}}_{f(a)} + \sum_{i=1}^{L}\sum_{j=1}^{N} |\underbrace{r_{ij}(t+1)}_{f(a)} - \underbrace{r_{ij}(t)}_{f(s)}| \cdot m_{ij} \\
&\underbrace{+\mathbf{1}_{\{\sum_{i=1}^{L} r_{ij}(t+1)\geq 1\}} \cdot (\sum_{i=1}^{L} r_{ij}(t+1) - 1) + (1 - \mathbf{1}_{\{\sum_{i=1}^{L} r_{ij}(t+1)\geq 1\}}) \cdot 0}_{f(s')} = \\
&C(s,a,s')
\end{aligned}
$$

Results. Here, we have formulated an unconstrained RL problem, and our approach allows us to capture a variety of different scenarios, where the agent focuses more on (1), (2), (3), and/or a convex combination of the above. The question we need to answer through the results, is whether the RL algorithm finds the optimal solution of the problem conditioned on some combination of weights $\{w_1, w_2, w_3\}$, for a variety of different slice traffic demand patterns.

For simplicity, we will assume two types of slices that take discrete values, with $D = \{0, 1\}$ (this corresponds to an ON/OFF type of traffic) and we will assume that our slice demands can be either:

- IID: each time step, the traffic of the slice is random, independent, and comes out of the same distribution.
- Bursty: the traffic at time $t+1$ is random, but it is highly likely to be similar to t.

We test the algorithm in a scenario where we have $|L| = 3$ slices (with one VNF each) and $|N| = 3$ server nodes; each server node has a capacity of one unit. For these scenarios, we simulate traffic that is random, and an allocator/agent orchestrates in real time the traffic and assigns it to appropriate server nodes. The ultimate goal is to design an agent that manages to find the optimal policy in the initially unknown environment, that is "minimize the expected cost of time step". To assess whether the algorithm performs well/reasonably we simulate the offline MDP that is fully aware of the *system's stochastic dynamics*, and plot its expected per time step reward. In the following plots, we showcase two scenarios:

- Two slices are random, and one is bursty,
- Two slices are bursty and one is random.

The results justify that such approaches are able to learn the dynamics, and exploit simultaneously.

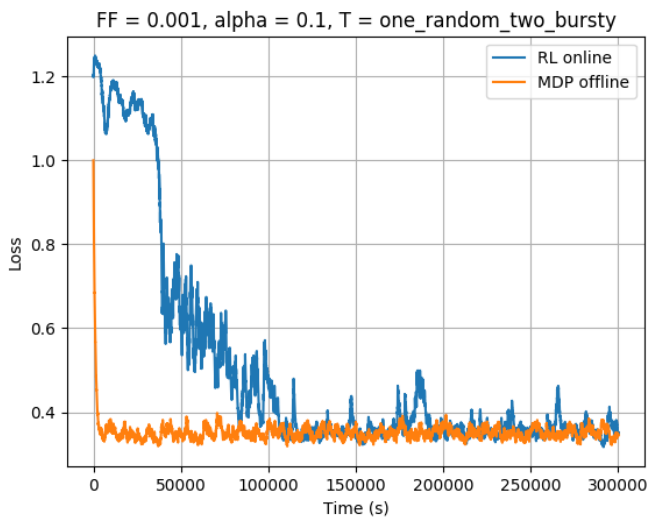


Figure 30 Objective performance vs time of online Q-learning and offline MDP-optimal controllers; case of one random and two bursty slices

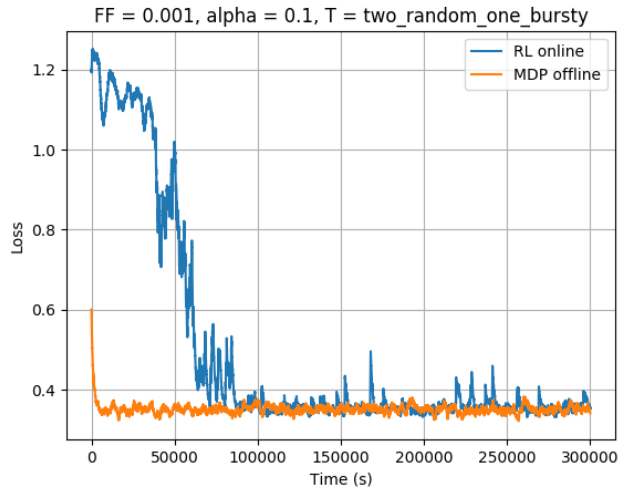


Figure 31 Objective performance vs time of online Q-learning and offline MDP-optimal controllers; case of two random and one bursty slices

Some conclusions. As we can readily see from the plots above, the RL-online algorithm can reach the optimal cost performance. However, the optimality comes at some cost. The time we observe at the x-axis is essentially time-slots, and as we can see, we need about 10^5 time instances until we reach the optimal performance. If each time instance was a second, this would be about 2.5 hours. Note that this is the time needed for a small-scale scenario, and this would probably further increase with the increase of the example.

7.1.2 *STATISTICAL METHODS FOR VNF BOTTLENECK LOCALIZATION*

Model and notation

The concept of connecting multiple VNFs is an ordered chain to afford an end-to-end service with specific QoS requirements is known as Service Function Chaining (SFC). The SFC is complementary to the NFV forwarding graph concept since it enables the interconnection of multiple virtual functions or service functions over multiple domains in a dynamic and flexible way. The SFC architecture can be built on top of the NFV ETSI model. The latter model supports the management operations for the VNFs to deploy the network services from the service forwarding graphs and needed interfaces for the VNFs and lifecycle management. These features enable the implementation of the SFC concept taking into consideration the NFV standards specifications.

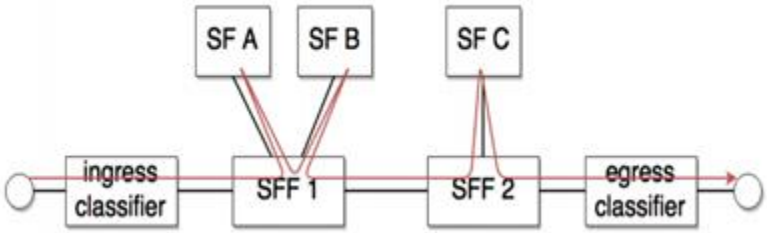


Figure 32 Service Function Chaining architecture

Explanation of Figure 32**:** It depicts the main components of the SFC architecture. The first element in the service chain is the Ingress Classifier, which is responsible for applying the traffic steering policy for matching the incoming flows with the needed Service Functions (SFs) where the packets can have specific processing. The SFs can be for example a firewall or a Deep Packet Inspection (DPI). A Service Function corresponds to a VNF (or multiple connected VNFs) in the NFV MANO architecture. Finally, the Service Function Forwarder (SFF) is responsible for forwarding the traffic to the SFs or to the next SFFs according to the SFC encapsulation information. The path taken by a packet formed by the SFFs and SFs is called the Service Function Path (SFP).

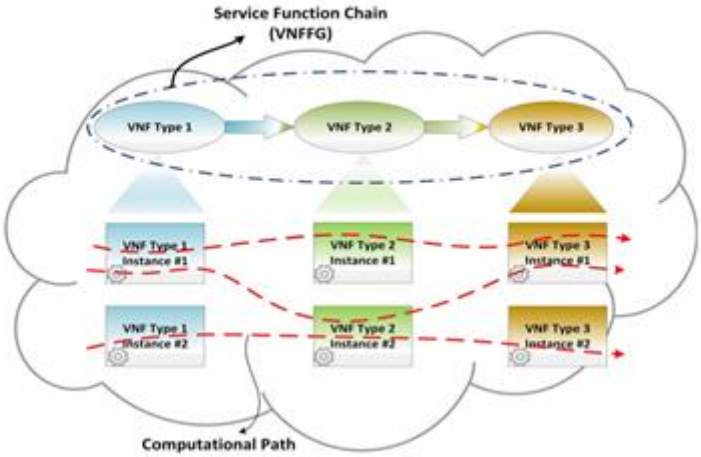


Figure 33 Service function chain deployment with multiple Service Function Paths

Explanation of Figure 33**:** It shows an example of an SFC deployment over an NFVI with multiple SFPs. For each SFP, a tunnel is created between the needed VNFs instances.

A *failed instance* can lead to disrupt all the requests passing through. It is possible to infer the intermediate nodes state from end-to-end measurements. There are some required conditions to fulfil while designing the

probing system to ensure the efficiency of such methods. These methods can be evaluated by its capacity to localize accurately the maximum of simultaneously failed points.

The monitoring system can detect/predict a performance degradation on an end-to-end path. In this case, the decision system must take remediation actions to correct the problem. The remediation action can be for example adding resources for the bottleneck nodes. End-to-end measures do not allow to localize these nodes accurately, so the decision system can deploy more sophisticated probing schemes to find them with minimum cost.

Decision Engine for probing strategy

We propose in this section an adaptive approach to select the needed set of probing paths. The first step of the monitoring operation is to cover the set of nodes V with a minimum number of paths from an exhaustive list denoted by P_{global}. This enables the detection of any misbehavior in the network topology. The process is described in lines 2-5 of Algorithm 1. It is based on an iterative process where we select at each iteration the path p that covers the maximum number of new nodes denoted by V_p until covering all the set of nodes noted V. The performed end-to-end measurements on this initial set of paths allow the detection of any anomaly in the network. Then, we need additional paths for the localization. The process of probing path selection for faulty nodes localization is a particular case of Group Testing Theory [72]. We follow an adaptive incremental method, where the DE selects at each iteration the path that gives the maximum of useful information according to the already measured paths. In our path selection strategy, we privilege those who are likely in an “UP” state.

Indeed, if the end-to-end measurement on a path is "UP”, we can conclude that all the nodes composing it are functional, which is not the case if the path is faulty. Thus, we compute at each iteration a score for the non-selected paths to choose the best one. This score reflects the expectation of the number of "UP” nodes for which we do not know their state weighted by the probability that the path is "UP”. This estimation is computed with formula:

$$E(p) = \left(\sum_{p \in V'} \alpha(0,p)\right) \text{Prob}(p_{state} = 0)$$

Where V' denotes the nodes with unknown states. At each iteration, the value of α is computed with the Algorithm inspired by [73].

INPUT: $\mathcal{G}$, $\mathcal{V}$, P_{global}
OUTPUT: $P_{selected}$.

$P_{selected} \leftarrow \emptyset$
while $\mathcal{V}$ is not covered **do**
  Select $p \in P_{global} | V^p \geq V^{p'}, \forall p' \in P_{global}$
  $P_{selected}.add(p)$
  $P_{global}.remove(p)$
end while
$Y \leftarrow getMeasurement(P_{selected})$
if $\exists p \in P_{selected} | Y^p = 1$ **then**
  while Failures are not localized **do**
    Initialize α
    $\mathbb{E} \leftarrow upNodesEstimates(P_{global}, \alpha)$
    $p^{max} \leftarrow p \in P_{global} | \mathbb{E}(p) \geq \mathbb{E}(p'), \forall p' \in P_{global}$
    $P_{selected}.add(p^{max})$
    $P_{global}.remove(p^{max})$
    $\alpha \leftarrow ESA(P_{selected})$
  end while
end if
return α

Algorithm: Probing path selection

Results

We use simulations to evaluate the proposed solution. Consider the case of a network service chain composed of 4 VNFs. Each network function has between 4 and 5 instances. We build a topology that respects the sufficient condition to localize *k* failures, and *k* is fixed to 1 in these tests.

The number of failed nodes is varied from 1 to 3. Figure 34 (below) illustrates the probability dispersion and the confusion matrix for the node states made with 50 tests. For each test, the algorithm gives the probability of failure on each node. These values give together the probability dispersion graph. Then, these values are compared to a fixed threshold to decide if the node is down or not to give the confusion matrix. With only one failure node, the monitoring system can localize them accurately in all the tests. These results are expected since the sufficient condition is satisfied to localize one failed node. However, this condition does not hold when more than one failure are generated as shown in the next tests in Figure 34. With two and three failed nodes, the inference algorithm identifies all of them. However, the number of false positives increases a little bit to reach 3% and 9% respectively.

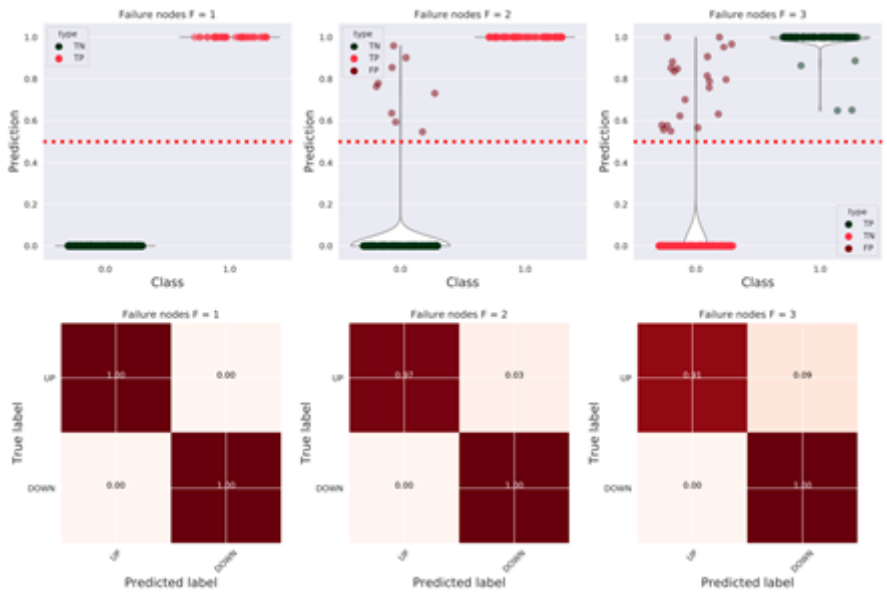


Figure 34 Failure probability estimations and confusion matrix for 50 tests. In the tests, we vary the number of simultaneously failed nodes from 1 tot3

7.1.3 LATENCY CONTROL

In the context of multi slice orchestration in the Radio Access Network (RAN) domain, radio resources are divided among the instantiated slices by the slice controller. The great variety of services to be instantiated translates in the need of accurate resource allocation schemes, and inter-slice isolation aspects to support the coexistence of latency-constrained, and throughput-constrained services. To fulfil this need, we design an orchestration solution that *autonomously* assigns chunks of radio spectrum (slices), hence pursuing the goal of simultaneously guaranteeing latency and throughput constraints.

When dealing with the RAN domain, the behavioural dynamics of the (aggregated) demand across involved tenants, and the inherent randomness of the wireless channel must be considered. Our solution takes into consideration the latter two aspects with a novel learning scheme which is able to extrapolate the implications of allocation decisions *on per-slice latency*, without explicitly making assumptions on the underlying dynamics. To this aim, we model our decision-making problem as a Markov Decision Process (MDP) to neglect low-level details of the tenant demands and channel dynamics. While to deal with state transition matrix statistics estimation, we rely to a Multi-Armed Bandit (MAB).

The Latency Control Problem

Let us define a network running slice $i \in I$ and the end-user $u \in U_i$ associated to the i-th slice. The total amount of wireless resources is split into multiple non-overlapping network slices. Based on fixed SLAs, each network slice is characterized by maximum throughput and expected latency denoted by Λ_i and Δ_i, respectively. Each

base station (BS) is characterized by a capacity C, thus the spectrum assignment per each slice, denoted as y_i, must satisfy the capacity constraint $\sum_{i\in I} y_i \leq C$.

We consider a time-slotted system where time is divided into decision epochs $n = \{1, 2, \ldots, N\}$, with duration ϵ that may be decided according to the infrastructure provider policies. We define the experienced instantaneous signal-to-noise ratio (SNR) of slice i (averaged over all $u \in U_i$) and the instantaneous aggregate traffic demand within time-slot n as Random Variables (RV)s $\gamma_i^{(n)}$ and $\lambda_i^{(n)}$, respectively. Our system does not assume any knowledge on such random variables, and exploits machine learning techniques to overcome this limitation.

The control problem becomes:

$$\min \lim_{N\to\infty} \sum_{n=1}^{N} \mathbb{E}\left[\sum_{i\in I} r_i^{(n)}\right]$$

$$s.t.\ \mathbb{E}\left[\frac{\lambda_i^{(n)}}{\varsigma\left(y_i^{(n)}, \gamma_i^{(n)}\right) + r_i^{(n)}}\right] \leq \Delta_i, \forall i \in I$$

$$\sum_i y_i^{(n)} \geq C, \forall n$$

$$y_i^{(n)}, r_i^{(n)} \in \mathbb{Z}_+, \forall i \in I$$

where $\varsigma(\cdot)^{(n)}$ is a mapping function that returns the number of bits that can be served using the allocated number of PRBs ($y_i^{(n)}$) and the current SNR level ($\gamma_i^{(n)}$), the agreed slice latency tolerance Δ_i, and $r_i^{(n)}$ introduced to account for the additional delay due to packet queuing during the time-slot n. To address this optimization problem, we rely on a two-layer scheduling approach wherein an inter-slice scheduler is in charge of defining the Physical Resource Blocks (PRBs) allocation strategy, and an intra-slice scheduler enforces the assignment of the pre-allocated subset of PRBs to the connected end-users. Our work mainly focuses on the higher-level inter-slice scheduler, leaving the implementation of intra-slice scheduling strategies open to address tenant-specific requirements. To solve the PRB allocation problem, we propose a two-stage approach that models channel and traffic demand variations based on previous observations and iteratively applies slice settings towards the goal of honouring SLAs.

Discrete Time Markov Chain Model

Expected channel conditions and violations on latency tolerance are analyzed through a Discrete Time Markov Chain (DTMC) model that accounts for the system dynamics over each slice, as depicted in Figure 35. We consider Markov chain time-slot that is *smaller* than the channel coherence time, in order to model variations as a sequential visiting of consecutive states.

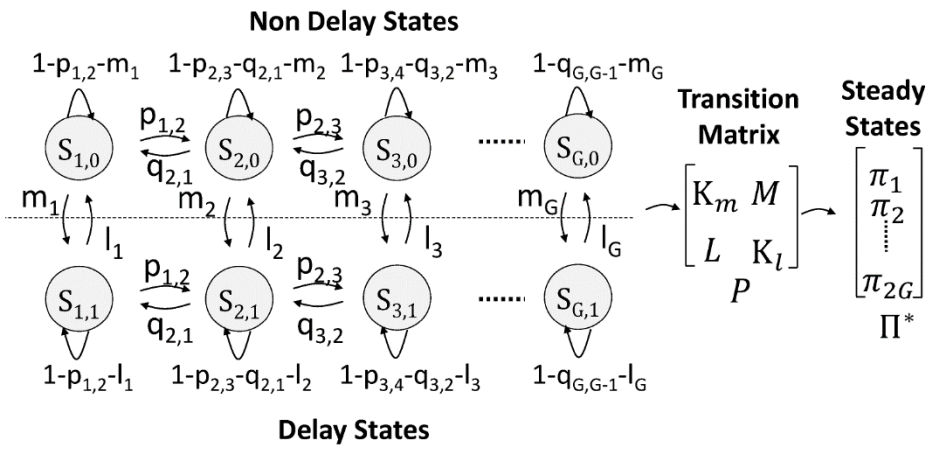


Figure 35 Radio channel variations as Markov chain.

For the sake of tractability, we consider an aggregate channel condition resulting from the set of users $u_i \in U_i$ belonging to slice i, and bound the channel quality to a finite set of quality levels G. Let us consider a discrete-time stochastic process X_t that takes values from a finite and discrete state space denoted by $S=\{S_0, 0, \ldots, S_{g,d}, \ldots, S_{G,1} \mid 0 \leq g \leq G, d \in \{0, 1\}\}$. Visiting state $S_{g,d}$ represents (a): an experienced channel level $g \in G$, with (b): an associated delay exceeding the one specified by the slice SLA ($d = 1$) or otherwise ($d = 0$).

Hence, we define the probability to improve (or worsen) the user channel condition from level g to level $g + 1$($g - 1$) as $p_{g, g+1}$ ($p_{g, g-1}$). Last, we model the probability to experience delay constraint violation as m_g and the probability to keep the access delay within the agreed bound as l_g. Thus, this process can be formulated as a two-dimensional DTMC $M := (S, P)$, where P denotes the state transition probability.

To estimate the matrix P, we rely on unsupervised learning, and more specifically on the well-known theory of probabilistic latent variable. Let us consider $w \in W$ as the stochastic latent variable denoting the current channel quality. Formally, we can redefine the transition probability of the DTMC as $\rho_{a,b}^{g} = \Pr(X_t = S_{g,b} \mid X_{t-1} = S_{g,a}, g = w)$ that is the transition probability from state $S_{g,a}$ to $S_{g,b}$ with channel quality $g = w$. The weight of each latent variable based on a given set of previous observations can be evaluated as

$$\omega(w|\hat{S}_t) = \frac{\sum_{\{\alpha,\beta\}\in \hat{S}_t} \rho_{\alpha,\beta}^{w}}{\sum_{g\in W} \sum_{\{\alpha,\beta\}\in \hat{S}_t} \rho_{\alpha,\beta}^{g}}$$

where $\hat{S}_t$ denotes the history of transitions across X_t among different states, and $\{\alpha, \beta\} \in \{0,1\}^2$. We can generalize the probability to move from a state wherein the latency is under control $S_{g,0}$ to some state incurring unexpected latency $S_{g,1}$, using the following expression

$$\rho_{a,b} = Pr(X_{t+1} = S_b \mid X_t = S_a, \widehat{S}_\iota) = \sum_{w \in W} \omega(w|\widehat{S}_\iota)\rho_{a,b}^{w}$$

Markov Decision Process (MDP)

The decision problem is modelled as a Markov Decision Process (MDP) defined by the set of states $\sum=\{\rho\}$, the set of actions $\Phi=\{\phi\}$, the transition function $T(\rho, \phi, \sigma')$, and the reward function $R(\rho, \phi)$.

The set of states accounts for all the slicing configurations $c_\sigma = \{y_1,\ldots, y_I\}$ expressed in terms of PRBs, where $\sum_{i \in I} y_i \leq C$. The transition function characterizes the system dynamics state σ to state σ' through action ϕ. Finally, the function $R(\rho,\phi)$ measures the reward associated to the state transition, when performing action ϕ. The policy for the decision agent is defined as $P^{(n)} : \sum^{(n)} \rightarrow \Phi^{(n)}$ and specifies which action to perform at time n when the system is in state σ. The final aim of the decision agent is to find the policy that maximizes the expected total reward.

To maximize the reward, each slicing configuration is associated with a reward action which is linked to the probability of exceeding the latency constraints defined in the slice SLA. Given a slicing configuration $c_\sigma = \{y_i \mid i \in I\}$, if the associated transition probability matrix P is perfectly known, we can derive the steady-state probabilities $\Pi^* = \{\pi_s^*\}$; this can be used to formulate the instantaneous reward value $R(\sigma^{(n)}, \phi^{(n)}) = (\sum_{s \in S_{g,0}} \pi_s^*)^\eta$ where s is the index of all states $S_{g,0}\ \forall g \in G$ such that the slice latency is under control, whereas $\eta \in [0,1]$ is an adjustable value decided by the infrastructure provider to provide action fairness in the reward function when η tends to 0, or maximum likelihood of keeping latency under control when η tends to 1.

Our objective is to maximize the expected aggregate reward obtained as $\lim_{n\to\infty} \sum_{n=1}^{N} \mathbb{E}[\chi^n R(\sigma^{(n)}, \phi^{(n)})]$. To achieve this goal, we need to *rely on* the transition probabilities $\rho_{a,b}$ inferred *based on the previous observations*.

Multi armed bandit problem

The proposed MDP can be solved by using dynamic programming solutions such as Value Iteration [16]. These approaches require exploring the entire state space of the MDP (several times) and the associated rewards. Let us consider a scenario with I online slices running in our system. Assume that each slice configuration y_i can take values from integer multiples of a minimum PRB chunk size Θ. Then, we can calculate the overall number of states equal to $(C/\Theta + I-1)! / ((I-1)!\ (C/\Theta)!)$. This poor state scalability, as well known as the curse of dimensionality, compromises the feasibility of MDP models under practical conditions.

However, MDPs provide insights regarding the structure of the problem itself and are very helpful to design auxiliary solutions, such as Multi-Armed Bandit (MAB) models, which are better suited for functional deployments. Therefore, in the next section we rely on a novel MAB design that exploits information from the underlying MDP to expedite the learning process while attaining near-optimal results.

Multi- Armed Bandit (MAB) problem emulates the action of iteratively select a single bandit (or slot machine) that may return the best payoff. Each slot machine returns unpredictable revenues out of fixed statistical distribution, not known a priori, that is iteratively inferred by previous observations. This matches well the randomness of the channel quality and the traffic demand we aim to capture whereas each bandit can be

mapped onto a slicing configuration state, with the final objective of maximizing the overall gain after a finite number of rounds. Let us define each arm $\sigma \in \sum$ as a different slicing configuration c_σ. Once selected, each arm provides an instantaneous reward $R(\sigma) = \sum_{i \in I} (\zeta(y_i, \gamma_i) - \lambda_i/\Delta_i)$ where the slicing configuration is $y_i \in c_\sigma$, $\zeta(\cdot)$ computes the number of bits that can be served using y_i configuration and given the current channel quality γ_i, and λ_i is the slice traffic demand. The bandit's reward is thus the expectation of access latency exceeding slice SLA.

Numerical Results

To assess heterogeneous slices, we simulate the network load demand of slice *i* at each time-slot (i.e., each transmission time interval (TTI) in Long Term Evolution (LTE) systems). Each slice has a normally distributed latency constraint, and Rayleigh channel is considered.

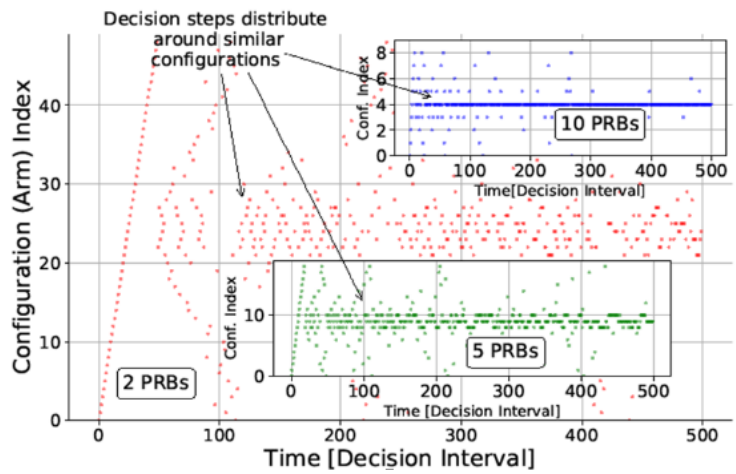


Figure 36 Impact of different resource allocation chunk sizes

Figure 36 shows the trade-off between the action space of the MAB agent (and its granularity) and the associated reward loss. To this aim, we set up a simple experiment with *2* slices with equal SLA requirements in a deterministic and static environment. 3 different action sets are available to the orchestrator: *{0, 2, 4, … , 100}*, *{0, 10, 20, … , 100}* and *{0, 10, 20, … , 100}* PRBs (with 50, 20 and 10 available configurations each), labelled "2 PRBs", "5 PRBs" and "10 PRBs", respectively. The results, shown in Figure 36, make it evident that the higher the granularity the longer the exploration phase(s): over 50 intervals for "2 PRBs", whereas it takes around 10 intervals for "10 PRBs". Interestingly, the loss in reward attained to the latter configuration is only 2%. Therefore, due to a faster convergence time at the expense of minimal reward loss, we empirically select 10 PRBs for our purposes.

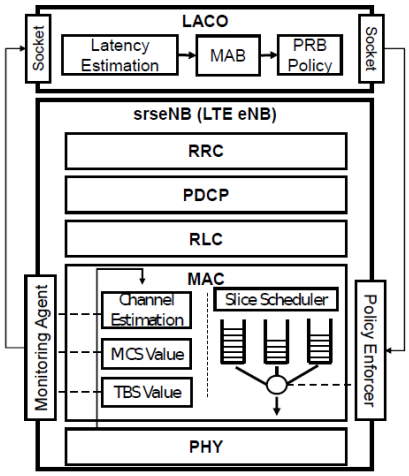


Figure 37 Architecture overview

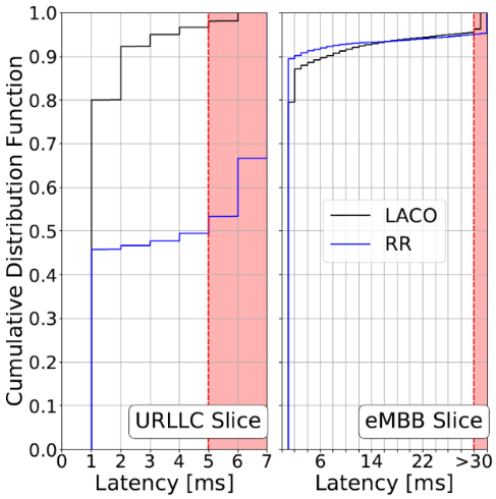


Figure 38 Preliminary results on Latency control

In order to illustrate, validate and analyze the performance of our LACO solution, we developed it as a standalone software module running on top of an open-source platform that implements the LTE protocol stack, namely srsLTE [74] and commercial tablets as UEs. The architecture of our software implementation and

LACO's interfaces with srseNB are depicted in Figure 37. LACO interacts with the eNB's Medium Access Control (MAC) layer to implement two key features: I) **Monitoring agent**. This feeds LACO with real-time SNR reports generated by the physical (PHY) layer from feedback received from the UEs, the selected MCSs and

corresponding transport block size (TBS) value used to encode information at the MAC layer, and other traffic statistics such as packet size and arrival times; II) **Policy Enforcer**. This allows LACO to dynamically enforce the PRB allocation policies calculated by our MAB model.

This information, together with the scheduling buffer size and data arrival times, is essential to compute the latency experienced by the different slices running in the system. To highlight the ability of dealing with heterogeneous slices, the empirical CDF of delay for an URLLC (5 ms) and eMBB (30 ms) slices sharing the radio interface is shown in Figure 38, against standard Round Robin (RR) policy. Results show a remarkable improvement of latency CDF for the URLLC slice, almost without affecting the performances of eMBB.

7.2 Cross-domain end-to-end inter-slice orchestration with Reinforcement Learning

For this level of controller, the decisions taken at one domain *do affect* the performance of the whole system. The end to end orchestrator has a wider view of the infrastructure and the analytics of the slices. As a result, it can capture more interesting tradeoffs, as the decisions on one technological domain may need to come with decisions on the other domain as well. Mathematically, the problem has similar complexity to the one of single (or domain specific) inter-slice orchestrator, however its state and action spaces are dramatically larger.

Slice and Infrastructure: Modelling and Problem Setup. In order to apply E2E centralized reinforcement learning control we first have to extend the model presented in subsection (7.1.1). This was a decoupled version of the network focusing only on the RAN domain. Here we aim to incorporate different network domains (RAN and Core) and consider an E2E metric. In a setup like that the actions taken for each of the domains are coupled. In this subsection we will mainly stress out the additions required on top of the existing model presented in (7.1.1).

We mainly utilize the same simple model for the network and the slices but we introduce some additional elements that will allow us to consider the E2E delay for each slice. Our system consists of N nodes on the RAN and M nodes on the Core Network. For the moment we assume that there is a total of $N \times M$ available links that connect RAN with Core nodes (one for each pair), and that each slice comes with one RAN VNF and one Core VNF, but this can be easily generalized. The nth RAN and m^{th} Core nodes have a total available capacity of C_n^R and C_m^C accordingly, while the capacity of the link that connects them is C_{nm}^L. Time is slotted since we consider again the same Time Scale as in section (7.1.1). We assume that there is a Service Level Agreement for each slice to be respected by the network operator. From the viewpoint of the enterprise requesting the slice, the quality of service could be perceived by the mean response time of the system or in other words the average E2E delay for a transmitted packet. Hence, we denote by T_i^{max} the maximum E2E delay according to the SLA for the ith slice. Also, we assume that the arrival of packets at each node or link is described by a Poisson process with rate λ. In the simple scenario of a slice that consists of one VNF on the RAN and one on the Core, the slice can be sufficiently described by λ_i^R, λ_i^C, λ_i^L, T_i^{max}, where λ_i^R, λ_i^C and λ_i^L are the arrival rates (or traffic demands) for the RAN VNF, the Core VNF and the link that connects them accordingly. Moreover, we assume that the service times at the nth RAN node, the mth Core node and the link between them are exponentially distributed with mean values $1/C_n^R$, $1/C_m^C$ and $1/C_{nm}^L$ respectively. Hence,

we can calculate the end-to-end delay of the system by modelling each node as well as each interconnection link by an M/M/1 queue. Since the number of packets arriving between reconfigurations of the system is very large, we can use the closed form expressions for M/M/1 queues to calculate the mean response time of the system (stationarity is required). Note that for a stable M/M/1 queue, Poisson arrivals with rate λ and exponential distributed service times with mean $1/\mu$, the mean response time is given by $1/(\mu-\lambda)$.

State Space. The configuration of the system can be represented by two indicator matrices, one for the RAN ($\boldsymbol{x}^R$) and one for the Core ($\boldsymbol{x}^C$) with size $L\times N$ and $L\times M$ respectively (i.e. if some capacity of the nth RAN node is allocated to the VNF of the ith slice we set $x_{in}^R=1$, otherwise it is $x_{in}^R=0$). The configuration of the RAN and Core implies also the configuration of the links (if the first VNF of the ith Slice runs on the nth RAN node and the second VNF runs on the mth CORE node then $x_{inm}^L=1$). Hence, for each slice there are $N\times M$ possible configurations in total. Also, each of the slice traffic demands is defined by a $D\times D$ transition probability matrix as remarked in (7.1.1), and there are $3L$ such matrices in total (3 for each slice). In such case, considering that a state is sufficiently defined by the RAN, Core configurations and the traffic demands, the total number of states is $N^L\times M^L\times D^{3L}$.

Action Space. The action we take at time t is the RAN and Core configurations at time $t+1$. Hence, at each state there are $N^L\times M^L$ possible actions.

Reward/cost. Here we use the same KPIs of interest as in subsection (7.1.1.). These are Free up Resources, Reconfiguration Costs and SLA Violations. However, we will formulate the SLA violations cost in a way that reflects the slice-tenant perceived experience and we will depart from the network-operator centred objective. Towards this direction we will use the E2E delay as a metric. Consequently, the objective now becomes:

$$
\begin{aligned}
C_{ss'}^{a} &= C_1(a) + C_2(s,a) + C_3(s') = \\
&= \underbrace{\sum_{n=1}^{N} \mathbf{1}_{\{\sum_{i=1}^{L} x_{in}^R(t+1)\geq 1\}} + \sum_{m=1}^{M} \mathbf{1}_{\{\sum_{i=1}^{L} x_{im}^C(t+1)\geq 1\}}}_{C_1} + \\
&+ \underbrace{1/2\cdot\sum_{i=1}^{L}\sum_{n=1}^{N} |x_{in}^R(t+1) - x_{in}^R(t)| + 1/2\cdot\sum_{i=1}^{L}\sum_{m=1}^{M} |x_{im}^C(t+1) - x_{im}^C(t)|}_{C_2} + \\
&+ \underbrace{\sum_{i=1}^{L} SLAv_i \cdot v_i^{ind}}_{C_3}
\end{aligned}
$$

Where:

$$
\begin{aligned}
SLAv_i &= T_i^{tot} - T_i^{max} = T_i^R + T_i^L + T_i^C - T_i^{max} = \\
&= \frac{1}{\sum_{n=1}^{N}[\ (C_n^R - \sum_{j=1}^{L} \lambda_j^R(t+1) \cdot x_{jn}^R(t+1)) \cdot x_{in}^R(t+1)]} + \\
&+ \frac{1}{\sum_{n=1}^{N} \sum_{m=1}^{M}[\ (C_{nm}^L - \sum_{j=1}^{L} \lambda_j^L(t+1) \cdot x_{jnm}^L(t+1)) \cdot x_{inm}^L(t+1)]} + \\
&+ \frac{1}{\sum_{m=1}^{M}[\ (C_m^C - \sum_{j=1}^{L} \lambda_j^C(t+1) \cdot x_{jm}^C(t+1)) \cdot x_{im}^C(t+1)]} - T_i^{max}
\end{aligned}
$$

And,

$$
v_i^{ind} = \begin{cases} 1 & \text{if } T_i^{tot} > T_i^{max} \\ 0 & \text{else} \end{cases}
$$

Let us see how this objective captures the slice viewpoint. The first two terms are no different than the corresponding terms presented in (7.1.1.). They are linear and capture the "Free up Resources" and "Reconfiguration costs" for RAN and Core. However, C_3 term is nonlinear and couples the decisions we make for RAN with the decisions we make for Core, since it is an E2E metric. We observe that with this model a violation of the service level agreement for a slice is possible even when the total arrival rate associated with any node or link does not exceed its total rate of service. Also, when a node or link is congested (the total arrival rate associated with this node or link gets close to its capacity), it acts like a bottleneck for the end-to-end transmission and affects all the associated slices. Note that this objective sums the SLA violations of each slice and not the under-provisioning of each node or link of the system. i.e. when there is a serious delay due to the overload of a specific node, this is taken into account as many times as the number of slices associated with this node and not just once. This is why it captures the SLA violation from the viewpoint of the slice and not from the viewpoint of the network operator. Also, note that the expressions for T_i^R, T_i^L and T_i^C hold only when the total arrival rate associated with any node or link does not exceed its total rate of service. Hence, in any other case we have to assign an infinite value to T_i^R, T_i^L, T_i^C. A last remark concerns the indicator variable v_i^{ind} of the C_3 term, which ensures that C_3 takes only positive values. Without that the algorithm would try to make this term as negative as possible leading to overprovisioning.

8 Conclusions

In this deliverable, we have presented the initial progress of the distributed AI-driven decision engine (DE), proposed in Work Package 4. We identified the key axes of novelty of the proposed mechanisms, mainly as the co-design of: (a) a stack of algorithms that is fully data-driven, and based on state-of-the-art machine learning methods that are revisited taking into account the intricacies specific to cellular networking setups, that force these algorithms to become network-aware or network-friendly; (b) a distributed implementation of such algorithms that is flexible enough to adapt from hierarchical to fully decentralized "federated" operation.

Specifically, in this deliverable we have discussed appropriate models and assumptions related to the problem setup(s) that are appropriate for the lifecycle management algorithms for beyond 5G slicing, such as slice representations, KPI metrics for the algorithms to optimize as well as KPIs used to measure the performance of the algorithms themselves, SLA types and violation penalties, etc. We have also provided an up-to-date state-of-the-art discussion on existing solutions, compared to the original discussion in the proposal. Based on the generic setup of the problem, we then proposed a flexibly distributable architecture for the DE, along with all the intra DE interfaces as well as interfaces with the MS and AE system. This architecture will be the basis for the integration and implementation envisioned in Task 4.4, to take place in the remainder of the project.

Finally, in the last three chapters, each pertaining to the main technical tasks of the project, namely tasks 4.1, 4.2, and 4.3, we have discussed in detail our initial attempts towards efficient algorithms related to the key phases of modern slice lifecycle management, namely admission control (Section 5), intra-slice management (Section 6), and inter-slice management (Section 7). There has been a range of methodology and AI-related tools investigated ranging from Deep Neural Networks adapted for slice resource allocation, to (tabular and deep) Reinforcement Learning methods (a natural candidate for control problems arising in both intra- and inter-slice management problems), even briefly touching upon "adversarial" (worst-case) methods that are a robust and fast-learning tool for highly non-stationary setups (which we plan to explore more in the remainder of the project).

In the remainder of the project, we will use the outcomes of this deliverable as the main guide towards achieving the following goals for the project: (a) we will investigate how the DE architecture proposed in Section. 4 can be integrated and demonstrated on an experimental platform, with multiple technical domains; (b) a large number of algorithms described in this deliverable were initially targeting a single domain; we plan to evolve these algorithms (or appropriately synthesize different components) towards full-fledged end-to-end solutions that span multiple technological and administrative domains; (c) we will further improve the individual algorithms as well as attempt to compare them in a common fair setup, with similar assumptions, and KPIs, in order to understand the (case-)specific pros and cons of our algorithmic arsenal.

9 References

[1] "NGMNAlliance,2016(Online). Available:, https://www.ngmn.org/wpcontent/uploads/160113_NGMN_Network_Slicing_v1_0.pdf".

[2] "3GPP Specification #23.501, System Architecture for the 5G System (5GS), 2018".

[3] "Network Functions Virtualisation (NFV) Release 3; Evolution and Ecosystem; Report on Network Slicing Support with ETSI NFV Architecture Framework, available: https://www.etsi.org/deliver/etsi_gr/NFV-EVE/001_099/012/03.01.01_60/gr_NFV-EVE012v030101p.pdf".

[4] "IETF Draft, Technology Independent Information Model for Network Slicing, available: https://www.ietf.org/archive/id/draft-qiang-coms-netslicing-information-model-02.txt".

[5] "https://private.matilda-5g.eu/documents/PublicDownload/119".

[6] NGMN Alliance, "5G White Paper," 2015.

[7] P. Vizarreta, M. Condoluci, C. M. Machuca, T. Mahmoodi and W. Kellerer, "QoS-driven function placement reducing expenditures in NFV deployments," in *2017 IEEE International Conference on Communications (ICC)*, 2017.

[8] D. Cotroneo, L. De Simone, A. K. Iannillo, A. Lanzaro, R. Natella, J. Fan and W. Ping, "Network Function Virtualization: Challenges and Directions for Reliability Assurance," in *2014 IEEE International Symposium on Software Reliability Engineering Workshops*, Naples, Italy, 2014.

[9] 5G PPP Architecture Working Group, "View on 5G Architecture," 5G PPP Architecture Working Group, 2020.

[10] M. O. Ojijo and O. E. Falowo, "A Survey on Slice Admission Control Strategies and Optimization Schemes in 5G Network," *IEEE Access,* vol. 8, pp. 14977-14990, 2020.

[11] S. E. Elayoubi, S. B. Jemaa, Z. Altman and A. Galindo-Serrano, "5G RAN Slicing for Verticals: Enablers and Challenges," *IEEE Communications Magazine,* vol. 57, no. 1, pp. 28-34, 2019.

[12] Y. L. Lee, J. Loo, T. C. Chuah and L. Wang, "Dynamic Network Slicing for Multitenant Heterogeneous Cloud Radio Access Networks," *IEEE Transactions on Wireless Communications,* vol. 17, no. 4, pp. 2146-2161, 2018.

[13] S. Vassilaras, L. Gkatzikis, N. Liakopoulos, I. N. Stiakogiannakis, M. Qi, L. Shi, L. Liu, M. Debbah and G. S. Paschos, "The Algorithmic Aspects of Network Slicing," *IEEE Communications Magazine,* vol. 55, no. 8, pp. 112-119, 2017.

[14] B. Han, D. Feng and H. D. Schotten, "A Markov Model of Slice Admission Control," *IEEE Networking Letters,* vol. 1, no. 1, pp. 2-5, 2019.

[15] A. Banchs, G. de Veciana, V. Sciancalepore and X. Costa-Perez, "Resource Allocation for Network Slicing in Mobile Networks," *IEEE Access,* vol. 8, pp. 214696-214706, 2020.

[16] T. Kuo, B. Liou, K. C. Lin and M. Tsai, "Deploying chains of virtual network functions: On the relation between link and server usage," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, 2016.

[17] Q. Zhang, F. Liu and C. Zeng, "Adaptive Interference-Aware VNF Placement for Service-Customized 5G Network Slices," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019.

[18] M. A. Tahmasbi Nejad, S. Parsaeefard, M. A. Maddah-Ali, T. Mahmoodi and B. H. Khalaj, "vSPACE: VNF Simultaneous Placement, Admission Control and Embedding," *IEEE Journal on Selected Areas in Communications,* vol. 36, no. 3, pp. 542-557, 2018.

[19] A. Fendt, S. Lohmuller, L. C. Schmelz and B. Bauer, "A Network Slice Resource Allocation and Optimization Model for End-to-End Mobile Networks," in *2018 IEEE 5G World Forum (5GWF)*, 2018.

[20] A. Othman and N. A. Nayan, "Efficient admission control and resource allocation mechanisms for public safety communications over 5G network slice," *Telecommun. Syst.,* vol. 72, pp. 595-607, 2019.

[21] J. Salvat, L. Zanzi, A. Garcia-Saavedra, V. Sciancalepore and X. Costa-Perez, "Overbooking Network Slices through Yield-Driven End-to-End Orchestration," in *Proceedings of the 14th International Conference on Emerging Networking Experiments and Technologies*, New York, NY, USA, 2018.

[22] B. Han, V. Sciancalepore, D. Feng, X. Costa-Perez and H. D. Schotten, "A Utility-Driven Multi-Queue Admission Control Solution for Network Slicing," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019.

[23] B. Han, J. Lianghai and H. D. Schotten, "Slice as an Evolutionary Service: Genetic Optimization for Inter-Slice Resource Management in 5G Networks," *IEEE Access,* vol. 6, pp. 33137-33147, 2018.

[24] P. T. A. Quang, Y. Hadjadj-Aoul and A. Outtagarts, "A Deep Reinforcement Learning Approach for VNF Forwarding Graph Embedding," *IEEE Transactions on Network and Service Management,* vol. 16, no. 4, pp. 1318-1331, 2019.

[25] M. Bunyakitanon, A. P. da Silva, X. Vasilakos, R. Nejabati and D. Simeonidou, "Auto-3P: An autonomous VNF performance prediction and placement framework based on machine learning," *Computer Networks,* vol. 181, 2020.

[26] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, K. Samdanis and X. Costa-Perez, "Optimising 5G infrastructure markets: The business of network slicing," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, 2017.

[27] T. V. K. Buyakar, H. Agarwal, B. R. Tamma and A. A. Franklin, "Resource Allocation with Admission Control for GBR and Delay QoS in 5G Network Slices," in *2020 International Conference on COMmunication Systems NETworkS (COMSNETS)*, 2020.

[28] V. Sciancalepore, X. Costa-Perez and A. Banchs, "RL-NSB: Reinforcement Learning-Based 5G Network Slice Broker," in *IEEE/ACM Transactions on Networking*, 2019.

[29] C. Song, M. Zhang, X. Huang, Y. Zhan, D. Wang, M. Liu and Y. Rong, "Machine Learning Enabling Traffic-Aware Dynamic Slicing for 5G Optical Transport Networks," in *2018 Conference on Lasers and Electro-Optics (CLEO)*, 2018.

[30] N. Salhab, R. Rahim, R. Langar and R. Boutaba, "Machine Learning Based Resource Orchestration for 5G Network Slices," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019.

[31] J. Kuo, S. Shen, H. Kang, D. Yang, M. Tsai and W. Chen, "Service chain embedding with maximum flow in software defined network and application to the next-generation cellular network architecture," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, 2017.

[32] G. Wang, G. Feng, W. Tan, S. Qin, R. Wen and S. Sun, "Resource Allocation for Network Slices in 5G with Network Resource Pricing," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, 2017.

[33] L. Gao and G. N. Rouskas, "On Congestion Minimization for Service Chain Routing Problems," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019.

[34] C. Marquez, M. Gramaglia, M. Fiore, A. Banchs and X. Costa-Perez, "How Should I Slice My Network? A Multi-Service Empirical Evaluation of Resource Sharing Efficiency," in *MobiCom '18*, New Delhi, India, 2018.

[35] C. Zhang, M. Fiore, C. Ziemlicki and P. Patras, "Microscope: Mobile Service Traffic Decomposition for Network Slicing as a Service," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, London, United Kingdom, 2020.

[36] D. Bega, M. Gramaglia, M. Fiore, A. Banchs and X. Costa-Pérez, "DeepCog: Optimizing Resource Provisioning in Network Slicing With AI-Based Capacity Forecasting," *IEEE Journal on Selected Areas in Communications,* vol. 38, no. 2, pp. {361-376}, 2020.

[37] H. A. Shah and L. Zhao, "Multiagent Deep-Reinforcement-Learning-Based Virtual Resource Allocation Through Network Function Virtualization in Internet of Things," *IEEE Internet of Things Journal,* vol. 8, no. 5, pp. 3410-3421, 2021.

[38] P. T. A. Quang, A. Bradai, K. D. Singh and Y. Hadjadj-Aoul, "Multi-domain non-cooperative VNF-FG embedding: A deep reinforcement learning approach," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2019.

[39] S. Messaoud, A. Bradai, O. Ben Ahmed, P. Quang, M. Atri and M. S. Hossain, "Deep Federated Q-Learning-based Network Slicing for Industrial IoT," *IEEE Transactions on Industrial Informatics,* 2020.

[40] Y. Sun, G. Feng, L. Zhang, P. V. Klaine, M. A. Iinran and Y. -C. Liang, "Distributed Learning Based Handoff Mechanism for Radio Access Network Slicing with Data Sharing," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019.

[41] E. M. D. R. S. H. a. B. A. y. A. H. B. McMahan, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*, 2017.

[42] A. S. A. T. a. V. S. T. Li, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine,* vol. 37, no. 3, pp. 50-60, 2020.

[43] C. C. M. S. a. A. T. V. Smith, "Federated multi-task learning," in *Advances in Neural Information Processing Systems*, 2017.

[44] C. L. J. K. U. E. a. D. S. N. Carlini, "The Secret Sharer: Measuring Unintended Neural Network Memorization & Extracting Secrets author={Nicholas Carlini and Chang Liu and J. Kos and {\'U}lfar Erlingsson and D. Song}, journal={ArXiv}, year={2018}, v," in *ArXiv*.

[45] A. N. a. A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control,* vol. 54, no. 1, pp. 48-61, 2009.

[46] C. R. S. W. a. F. N. B. Recht, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. Advances in neural information processing systems," *Advances in neural information processing systems,* pp. 693-701, 2011.

[47] A. O. a. W. S. A. Nedic, "Achieving geometric convergence for distributed optimization over time-varying graphs," *SIAM Journal on Optimization,* vol. 27, no. 4, pp. 2597-2633, 2017.

[48] K. Y. Q. L. W. Y. a. A. S. T. Wu, "Decentralized consensus optimization with asynchrony and delays," *IEEE Transactions on Signal and Information Processing over Networks,* vol. 4, no. 2, pp. 293-307, 2017.

[49] W. Z. C. Z. a. J. L. X. Lian, "Asynchronous decentralized parallel stochastic gradient descent," in *International Conference on Machine Learning*, July, 2018.

[50] W. H. a. F. I. P. Bianchi, "A coordinate descent primal-dual algorithm and application to distributed asynchronous optimization," *IEEE Transactions on Automatic Control,* vol. 61, no. 10, pp. 2947-2957, 2015.

[51] S. K. a. M. J. T. Vogels, "Powersgd: Practical low-rank gradient compression for distributed optimization," in *Advances in Neural Information Processing Systems*, 2019.

[52] J.-B. C. a. M. J. S. U. Stich, "Stich, S. U., Cordonnier, J.-B., and Jaggi, M. (2018). Sparsified SGD with memory. InAdvances in Neural Information Processing Systems," in *Advances in Neural Information Processing Systems*, 2018.

[53] S. S. a. M. J. A. Koloskova, "Decentralized Stochastic Optimization and Gossip Algorithms with Compressed Communication," in *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[54] J. Cao, Y. Zhang, W. An, X. Chen, J. Sun and Y. Han, "VNF-FG design and VNF placement for 5G mobile networks," *Science China Information Sciences,* vol. 60, no. 4, pp. 1869-1919, 2017.

[55] Z. Yan, J. Ge, Y. Wu, L. Li and T. Li, "Automatic Virtual Network Embedding: A Deep Reinforcement Learning Approach With Graph Convolutional Networks," *IEEE Journal on Selected Areas in Communications,* vol. 38, no. 6, pp. 1040-1057, 2020.

[56] H. Wang, Y. Wu, G. Min and W. Miao, "A Graph Neural Network-based Digital Twin for Network Slicing Management," *IEEE Transactions on Industrial Informatics,* 2020.

[57] A. Rkhami, Y. Hadjadj-Aoul and A. Outtagarts, "Learn to improve: A novel deep reinforcement learning approach for beyond 5G network slicing," in *IEEE Consumer Communications \& Networking Conference (CCNC)*, 2021.

[58] M. Schlichtkrull, T. Kipf, P. Bloem, R. van den Berg, I. Titov and M. Welling, "Modeling Relational Data with Graph Convolutional Networks," in *European Semantic Web Conference*, 2018.

[59] "ETSI ZSM 009-1, “Zero-Touch Network and Service Management (ZSM); Closed-loop Automation; Enablers”, V0.10.5 (2021-01)".

[60] Y. W. A. T. J. H. P. A. a. I. M. Ryan Lowe, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17)*.

[61] GSMA, "Generic Network Slice Template Version 4.0," 2020.

[62] Y. Chen, Y. Gao, C. Jiang and K. J. R. Liu, "Game theoretic Markov decision processes for optimal decision making in social systems," in *2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, Atlanta, GA, USA, 2014.

[63] P. D. Pandey, "Approximate Q-Learning: An Introduction," in *Second International Conference on Machine Learning*, Bangalore, 2010.

[64] D. B. a. A. V. A. Jeerige, ""Comparison of Deep Reinforcement Learning Approaches for Intelligent Game Playing,"," in *IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA, 2019.

[65] Q. Yu, J. Chen, Y. Sun, Y. Fan and X. Shen, "Regret Matching Based Channel Assignment for Wireless Sensor Networks," in *2010 IEEE International Conference on Communications*, Cape Town, South Africa, 2010.

[66] J. J. Alves Esteves, A. Boubendir, F. Guillemin and P. Sens, "Heuristic for Edge-enabled Network Slicing Optimization using the “Power of Two Choices”," in *16th International Conference on Network and Service Management (CNSM)*, 2020.

[67] R. A. L. C. Nguyen DD, "Online versus offline reinforcement learning for false target control against known threat.," in *Intelligent Robotics and Applications. ICIRA 2018.*.

[68] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *In International conference on machine learning*, 2016.

[69] Z. J. G. Y. W. L. L. a. T. L. Yan, "Automatic virtual network embedding: A deep reinforcement learning approach with graph convolutional networks," *IEEE Journal on Selected Areas in Communications,* vol. 38, no. 6, pp. 1040-1057, 2020.

[70] Ltd LM Ericsson, ""Methods and Apparatus for Managing a System that Controls an Environment"". US Patent 63/058985.

[71] D. M. G. M. F. A. B. a. X. C.-P. Bega, "AZTEC: Anticipatory capacity allocation for zero-touch network slicing," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, 2020.

[72] M. Aldridge, O. Johnson and J. Scarlett, "Group Testing: An Information Theory Perspective," *Foundations and Trends in Information Theory,* 2019.

[73] M. Rahali, J. Sanner and G. Rubino, "Unicast Inference of Additive Metrics in General Network Topologies," in *27th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS*, 2019.

[74] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano and D. J. Leith, "srsLTE: An open-source platform for LTE evolution and experimentation," in *ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*, 2016.

[75] M. Agiwal, A. Roy and N. Saxena, "Next Generation 5G Wireless Networks: A Comprehensive Survey," *IEEE Communications Surveys & Tutorials,* vol. 18, no. 3, pp. 1617-1655, thirdquarter 2016.

[76] A. Ksentini and N. Nikaein, "Toward Enforcing Network Slicing on RAN: Flexibility and Resources Abstraction," *IEEE Communications Magazine,* vol. 55, no. 6, pp. 102-108, 2017.

[77] A. Ksentini, P. A. Frangoudis, A. PC and N. Nikaein, "Providing Low Latency Guarantees for Slicing-Ready 5G Systems via Two-Level MAC Scheduling," *IEEE Network,* vol. 32, no. 6, pp. 116-123, 2018.

[78] 3GPP, "Study on Management and Orchestration of Network Slicing for Next Generation Network," 2018.

[79] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, K. Samdanis and X. Costa-Perez, "Optimising 5G infrastructure markets: The business of network slicing," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, Atlanta, GA, USA, 2017.

[80] S. Bakri, P. A. Frangoudis and A. Ksentini, "Dynamic Slicing of RAN Resources for Heterogeneous Coexisting 5G Services," in *2019 IEEE Global Communications Conference (GLOBECOM)*, Waikoloa, HI, USA, 2019.

[81] W. Qiang and Z. Zhongli, "Reinforcement learning model, algorithms and its application," in *2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*, Jilin, China, 2011.

[82] J. X. Salvat, L. Zanzi, A. Garcia-Saavedra, V. Sciancalepore and X. Costa-Perez, "Overbooking network slices through yield-driven end-to-end orchestration," in *CoNEXT '18: Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*, 2018.

[83] A. Anand, G. de Veciana and S. Shakkottai, "Joint Scheduling of URLLC and eMBB Traffic in 5G Wireless Networks," *IEEE/ACM Transactions on Networking,* vol. 28, no. 2, pp. 477-490, 2020.

[84] M. Vincenzi, E. Lopez-Aguilera and E. Garcia-Villegas, "Maximizing Infrastructure Providers' Revenue Through Network Slicing in 5G," *IEEE Access,* vol. 7, 2019.

[85] M. R. Raza, C. Natalino, P. Öhlen, L. Wosinska and P. Monti, "Reinforcement Learning for Slicing in a 5G Flexible RAN," *Journal of Lightwave Technology,* vol. 37, no. 20, pp. 5161-5169, 2019.

[86] D. Pandey and P. Pandey, "Approximate Q-Learning: An Introduction," in *2010 Second International Conference on Machine Learning and Computing*, Bangalore, India, 2010.

[87] A. Jeerige, D. Bein and A. Verma, "Comparison of Deep Reinforcement Learning Approaches for Intelligent Game Playing," in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA, 2019.

[88] D. D. Nguyen, A. Rajagopalan and C.-C. Lim, "Online Versus Offline Reinforcement Learning for False Target Control Against Known Threat," in *ICIRA 2018: Intelligent Robotics and Applications*, 2018.

[89] J. J. Alves Esteves, A. Boubendir, F. Guillemin and P. Sens, "Heuristic for Edge-enabled Network Slicing Optimization using the "Power of Two Choices"," in *2020 16th International Conference on Network and Service Management (CNSM)*, Izmir, Turkey, 2020.

[90] Y. G. C. J. a. K. J. R. L. Y. Chen, ""Game theoretic Markov decision processes for optimal decision making in social systems," 2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP), Atlanta, GA, 2014.".

[91] O. E. F. M. Ojijo, "A Survey on Slice Admission Control Strategies and Optimization Schemes in 5G Network," *IEEE Access,* no. 19313469, pp. 14977-14990, 2020.

[92] "ETSI ZSM 009-1, "Zero-Touch Network and Service Management (ZSM); Closed-loop Automation; Enablers", V0.10.5 (2021-01)".